Detection of Midair Finger Tapping Gestures and Their Applications

September 2017

Hani Karam

Detection of Midair Finger Tapping Gestures and Their Applications

Graduate School of Systems and Information Engineering

University of Tsukuba

September 2017

Hani Karam

Abstract

With the recent advancements in touchscreen and various sensors technology, interaction with computers is starting to slowly shift from the keyboard and mouse paradigm to inputting commands directly with the body. This kind of interactions forms the basis of Natural User Interfaces. Numerous possibilities exist in Natural Interfaces, midair interaction being one of them.

Multiple approaches related to midair interactions have already been proposed. However, many of them are limited in terms of interactions, vocabulary and naturalness; some of them make the users wear extra hardware or can be tiring or difficult to utilize. Most of them lack a mechanism that allows users to control their beginning and end. In this study, we focus on creating midair interfaces that feel natural by replicating midair tapping, a gesture that humans are used to execute on a regular basis. Such a gesture can be chained multiple times or simultaneously executed from different fingers to enrich the interaction vocabulary, all while keeping the gestures easy to remember and perform.

The starting point of our research was the detection of an index finger tap using a depth camera. It aimed at overcoming previously developed unintuitive selection gestures by replacing them with a familiar finger tapping gesture. Thus, we developed the “Three Fingers Clicking Gesture”, where the users fold their ring and small fingers, extend their thumb and middle fingers, and execute taps with the index finger. Our system had a good reliability and did not need any training for the software to be able to recognize clicks.

However, we have noticed some limitations of that system. The users were only allowed to move their index finger, and as a result, the index was the only finger that was being used for taps. In addition, the posture was ergonomically poor. To improve the previous detection system, we introduced “Depth Click” where the users could keep all of their fingers extended in a neutral position; this allowed the detection of taps from all five fingers, and the ergonomics improved by having a more relaxed posture. Furthermore, we have separated a click gesture into “Click Down” and “Click Up”. A click was therefore defined as a Click Down followed by a Click Up. This gave the users a full control over the tapping gestures, and they could perform as slow or as fast as they desired. Additionally, the separation of a tap into two distinct gestures gave rise to the ability of cancelling a gesture, even after it has started.

While Depth Click was better than the previous detection system, it was still limited to a generic tap detection mechanism, and its neutral position was causing some tiredness in the hand. To overcome its limitations, we have developed “Xpli Tap”, where we have introduced a new relaxed posture for the neutral position. In addition, the new algorithm detects finger clicks explicitly by extracting the fingers positions when they are at their deepest point during a tap. This made retrieving the exact position of the tap gesture possible. When multiple fingers moved together, we used their positions to isolate the tapping finger by analyzing their motions and determining the one that traveled the biggest distance. Furthermore, we were able to distinguish three shapes for a tapping finger by examining its clicking position, and we were thus able to use the shape to increase the interaction vocabulary. We have developed a midair keyboard as an application to Xpli Tap.

Finally, we created “MultiX Click”, a method that makes simultaneous taps possible. We were able to mix multi-clicks with regular clicks, thus increasing the gesture vocabulary. We could detect simultaneous taps originating from both hands. We have created a midair keyboard that simulates the usage of the Shift, Control, and Alt modifier keys. It made it possible to input a modifier key with one hand and a regular key with the other. We have introduced detection zones to prevent overlapping of regular taps and multi-clicks. Furthermore, we were able to detect the concurrent taps originating from multiple fingers of a single hand. As an application, we implemented a midair piano where it was possible to input single notes and multiple ones such as chords.

In summary, natural interfaces are slowly making their way into mainstream usage, albeit in a limited way. Our research highlighted some previous problems in midair interactions and aimed at overcoming some limitations and improving their usability by eliminating the difficulty of their execution. Moreover, the conducted experiments have shown that our proposed algorithms are efficient and reliable. In the future, our techniques may be incorporated in devices such as smartphones and smartwatches equipped with adequate sensors and enrich their usage by allowing the users to perform midair gestures that are not limited by the restricted size of their screens.

Acknowledgments

I am deeply grateful to Professor Shin Takahashi, my thesis supervisor, for his many valuable suggestions, precise directions, and kind encouragement. I am also heartily thankful to Professor Jiro Tanaka, my previous thesis supervisor for his great support, multiple comments and rigorous guidance. I also greatly thank Professor Buntaro Shizuki, Professor Akihisa Ohya, Professor Jun Mitani and Professor Tomoo Inoue for their many useful comments on this research. I would particularly like to thank Professor Simona Vasilache for her enormous support, vast generosity and kind heart which made me feel at home. I am also thankful to all the members of the IPLAB, Interactive Programming Laboratory, University of Tsukuba, for giving me many opportunities to discuss my research with them and helping me in my experiments, especially Mr. Ryosuke Takada whose benevolent attitude and kindness were essential to my work. I am also grateful to my friends who were very supportive and encouraging.

I would like to extend my sincere gratitude, gratefulness and respect to Professor Hady Kahy; his guidance, vast support, numerous advices and immense generosity were invaluable to my success. Finally, I extend my sincere gratitude to my parents for all their efforts and their wishes.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Gestures are an important topic in Natural User Interfaces (NUI) and have been studied for over thirty years. Gestures lack a mechanism that allows users to control their beginning and end. Our research aims at detecting midair finger taps and making gestures interaction easier to use, and increasing the interaction vocabulary through the use of the familiar gesture of tapping. The starting point of our research was the detection of an index finger tap using a depth camera. It aimed at overcoming previously developed unintuitive selection gestures by replacing them with a familiar finger tapping gesture. Thus, we developed the Three Fingers Clicking Gesture, where the users fold their ring and small fingers, extend their thumb and middle fingers, and execute taps with the index finger. Our system had a good reliability and did not need any training for the software to be able to recognize taps.

However, we have noticed some limitations of that system. The users were only allowed to move their index finger, and as a result, the index was the only finger that was being used for taps. In addition, the posture was ergonomically poor. To improve the previous detection system, we introduced Depth Click where the users could keep all of their fingers extended in a neutral position; this allowed the detection of taps from all five fingers, and the ergonomics improved by having a more relaxed posture. Furthermore, we have separated a tap gesture into Click-Down and Click-Up. A click was therefore defined as a click-down followed by a click-up. This gave the users a full control over the tapping gestures, and they could perform as slow or as fast as they desired. Additionally, the separation of a tap into two distinct gestures gave rise to the ability of cancelling a gesture, even after it has started.

While Depth Click was better than the previous detection system, it was still limited to a generic tap detection mechanism, and its neutral position was causing some tiredness in the hand. We wanted to overcome its limitations. We have therefore developed Xpli Tap, where we have introduced a new relaxed posture for the neutral position. In addition, the new algorithm detects finger taps explicitly by extracting the fingers positions when they are at their deepest point during a tap. This made retrieving the exact position of the click gesture possible. When multiple fingers moved together, we used the positions to isolate the tapping finger by analyzing their motions and determining the one that traveled the biggest distance. Furthermore, we were able to distinguish three shapes for a tapping finger by examining its tapping position, and we were thus able to use the shape to increase the interaction vocabulary. We have developed a midair keyboard as an application to Xpli Tap.

Finally, we created MultiX Click, an algorithm that makes simultaneous taps possible. We were able to mix multi-clicks with regular clicks, thus increasing the gesture vocabulary. We also implemented a midair piano as an application to multi-clicks.

1.1 Dissertation organization

This thesis is organized as follows. In Chapter 2, we introduce two techniques for distinguishing midair finger taps which could be detected in 3D using a depth camera. Chapter 3 introduces an algorithm which overcomes previous limitations, adds precision to the detection and enriches the gestures vocabulary. Chapter 4 describes a technique that allows the detection of simultaneous multiple midair finger taps. Finally Chapter 5 provides conclusions.

Chapter 2

Finger Click Detection Using a Depth Camera

In this chapter, we describe two approaches for midair finger tap detection using a depth camera. The design of each system is described, including detailed explanation of the algorithms. We start by describing the benefits of finger tap detection, then we conduct experiments to determine the usability and reliability of each approach. We then discuss the capabilities and limitations of the system. Finally, we talk about a prototype application of finger tap detection. Throughout the rest of the thesis, we will use the terms "tap" and "click" interchangeably to signify the same gesture.

2.1 Research Goals

Gestures have long been studied in the field of Human Computer Interaction (HCI). They form an important part of NUI, and their applications vary across a wide area, ranging from home appliances control [1, 2] to video games [3, 4] to interaction with medical devices in the operation room [5, 6]. Gesture data fetching is an important procedure in hand gesture recognition, and the used methods can be mainly divided into two categories [7]: glove-based and computer vision based. Using gloves forces the users to wear the sensors in order to have their gestures recognized. This can feel unnatural, as the ideal gestures in NUI should be performed by using just the body, without the need to wear any additional sensors. In such cases, cameras feel more natural than gloves. Cameras can be divided into two categories: RGB cameras and depth cameras. Computer vision is used with RGB cameras in order to recognize gestures. Vision based gestures identification is highly dependent on the environment; noise or a dark space can prevent gestures from being detected [8]. It also uses gestures that are mainly performed in 2D. In the recent years, advancements have made depth sensors widely available as commercial products, such as the SoftKinetic DepthSense and Leap Motion. Unlike RGB (color) cameras, depth cameras add an additional dimension in depth maps and thus allow data to be captured in 3D. Detection became more robust, and gestures can be performed in 3D thus increasing the interaction vocabulary. A lot of researches has been conducted on depth-based gesture recognition, however most of them feel unnatural and lack a precise way of controlling the start of the gesture identification. In this research, we would like to detect selection mechanism gestures that are natural for the users. They should also be intuitive and easy to perform. Furthermore, existing selection mechanism gestures lack the precision for controlling the selection to create a more interactive UI; adding a “tapping” option would make the interface more flexible and would give users the possibility to precisely select a command at will by granting them the opportunity to choose the exact timing of the command they wish to issue. Therefore, we propose new methods for detecting midair finger taps, since finger taps is a natural gesture. The gestures should be detectable regarding of the hand position in the 3D space. In order to increase the interaction vocabulary, they should have the capability of increasing the vocabulary from the existing basic selection.

2.2 Gesture Detection

Several gestures and gestures recognition techniques have been proposed [9], [10], [11], [12]. Gesture data fetching is an important procedure in hand gesture recognition, and the used methods can be mainly divided into two categories [7]: glove-based and computer vision based. A third, recently introduced approach can also be cited: depth-based data fetching. In this section, we describe some related work from each approach, while also discussing their limitations as well as the advantage of our approach over them.

2.2.1 Glove-based Hand Gestures Recognition

Data gloves use sensors embedded in gloves for digitizing hand and finger movements. They relay the digitized information to a computer system using wires. The sensors make the data captured by the gloves very precise, as they can capture the position of each finger, as well as the orientation of the hand. Therefore, using data-gloves for gesture recognition can result in high reliability and precision.

Lévesque et al. [10] presented a gestural interface that uses data gloves to increase efficiency and reliability. They were able to use multiple 3D gestures such as selecting and designating an object, then moving/rotating/resizing it. They were also able to define object-specific manipulations, as well as system control through menus.

Kenn et al. [13] used data gloves to create applications to a gesture based interface. They were able to get high recognition rate and provide an interesting user experience. Kumar et al. [14] used wireless data gloves using Bluetooth technology to detect various gestures such as left-clicking, right-clicking, dragging, rotating and pointing.

While data gloves can precisely acquire the hand gesture's data, they often encumber the user with tethers and setup time. Even wireless gloves are still bulky and not portable. "Come as you are" [15] is an HCI design paradigm that suggests that a system should be usable as soon as users are in the proximity of the system; they should be able to start using it as soon as they get close to it, without any additional setup. Data gloves do not comply with this paradigm because the users need to wear them first. Because of the wired environment, the users are limited in space and the overall apparatus might seem unnatural.

2.2.2 Vision-based Hand Gestures Recognition

Vision-based gesture recognition uses digital cameras usually connected to a computer system. Unlike data gloves (2.2.1) where hand information is relayed to the computer using wires, or forces users to wear gloves, vision based recognition is nonintrusive and untethered. This frees the users from donning any special hardware or apparatus to enable them to interact with the system, which gives rise to a more natural interaction [16].

Chu et al. [9] used a webcam and computer vision to create a hand gesture system for taking self-portaits. They detect the locations of the fingers by applying low-level image processing operations on each captured frame. To accomplish this, they first need to detect the hands. This is achieved by using skin-color detection algorithm [17], which categorizes every pixel as a skin-color pixel or non-skin color pixel. They then segment and apply contour finding algorithm to isolate the hands, after which they detect the fingertips by using curvature-based algorithms. In their interface, the users could use any of the detected fingers to interact with an onscreen button by extending this finger over it for one second.

As skin detection relies on finding pixel with color similar to that of the human skin, it is very sensitive to lighting conditions. The color can also be affected by some camera settings such as white balance. Due to this, skin and finger detection might not be optimal or even possible in some conditions. To overcome this, [18] used the Lucas-Kanade optical flow algorithm [19] which can be used to measure the motion gesture between two frames. This can be particularly useful if the hands move at high speed which results in a blurred image where fingertips detection might be impossible.

YCrCb color ranges was used in [20] for skin color extraction. They were able to detect faces and hands using this technique. Since their purpose is to detect hands, they had to perform a face removal step. They then execute a series of techniques such as contour extraction, polygon approximation and convex hull detection to determine the hand and fingertips locations and direction, which in turn were used to recognize simple hand gestures such as open palm and pointing.

Even though computer vision frees users from the tethers of data gloves, and complies with "Come as you are" paradigm, vision based recognition interactions can only be accomplished in 2D. Moreover, their success depends on many conditions such as proper lighting or slow hand movement speed. This makes the possible interactions limited to very specific environments and gestures.

2.2.3 Depth-based Hand Gestures Recognition

Unlike RGB based recognition which relies on analyzing features of a 2D image, depth-based recognition uses depth maps, which are images representing the depth (distance from the sensor). The introduction of Microsoft Kinect [21] has led to an increase in research on depth based recognition.

Dominio et al. [22] extract the hand from depth maps by also using information from the color image, and then segment it to palm and fingers regions. They then use two different set of feature descriptors, and employ a multi-class Support Vector Machines classifier to recognize gestures by classifying the vectors with the distance and curvature features concatenated into classes corresponding to the various gestures of the database.

In [23] the authors applied machine learning algorithms to detect gestures such as arm-swings, arm-push, zoom-in and zoom-out. They captured the depth data using a Kinect sensor and extracted features that were then fed to a nearest neighbor classifier. They succeeded in implementing a robust gesture recognition system.

Biswas et al. [24] first isolate a human from the background scene of a depth image by using auto-thresholding on the depth histogram. They then execute further operations on the resulting histogram to extract hands and determine regions of interest, which are then loaded in a multi-class SVM classifier which was used to train the system for the classification of the gestures.

The work in [25] uses a Kinect sensor to acquire depth images from which a hand's fingers are identified by applying several calculations on information retrieved from the hand. The fingers are then used in gestures recognition. Interesting interactions were then defined by using this approach.

2.3 Selection Mechanisms using Gestures

A User Interface (UI) is seldom made from just one item; it usually consists of several elements, such as buttons, menus, icons and drop-down lists. In a UI, each of these elements has its own use. To operate such an interface, a user has to be able to designate those elements distinctly, hence the need for selection mechanisms. In traditional Graphical User Interfaces (GUIs), users point to items (buttons, icons, etc.) by using an input device, usually a mouse, and select a given item by performing a click operation. However, in the recent years, with the advances in Natural User Interface (NUI), the research is mainly focusing on using hand gestures as a means for user interaction. Wachs et al. states that gestures tend to be an intuitive and natural way to communicate with a computer [8]. In [26], it is suggested that gestures can be an effective means of interaction with the everyday desktop. As gestures make a good candidate for controlling a computer system naturally, we therefore proposed using them as selection mechanisms in a NUI environment.

Gesture design is an important step in creating a NUI system. [8] suggest that gestures should be intuitive to users: when a gesture is performed, is should closely resemble the operation it is associated with. This helps users remember the gestures more easily. Moreover, the operation of the system will feel more natural. [27] suggest that "air tapping" (emulating a mouse click in midair) was ranked best form for midair interaction because of the familiarity with mouse clicking. In respect to that, we designed the selection mechanism gesture as a click.

2.3.1 Related Work

Chu et al. [9] used hand gestures to take self-portraits. They used an onscreen “hover button interface” as a selection mechanism, where they would hover with a finger to trigger the camera’s shutter. But because of the noise of detection and cluster background, they have to keep the finger for 1 second over the button. While this reduces the amount of errors, it introduces a waiting time in the interaction. After the time threshold has been crossed, the action is triggered. If the user wanted to cancel his action, he could only do so before the time trigger has passed, since the system does not provide the user with a flexible way to cancel a selection operation. The works [11] and [28] used a crossing technique that was originally developed for pen-based interfaces: the crossing event occurs when the cursor intersected with the boundary of a graphical object. While effective, these crossing techniques are limited to a basic selection operation. Another approach used in [10] consists of using “index pointing, thumb up” gesture to perform a selection. However, this gesture does not feel intuitive as a selection technique, and contradicts the idea presented in [8]. Liu et al. used postures which were defined by which fingers were raised or folded and whether the thumb was stretched out, aligned with the palm, or tucked into the palm [29]. Those postures used multiple fingers to enrich the vocabulary, and were used to issue command events or command parameters; however, they were not defined for precise input. Kulshreshth et al. introduced a 3D gesture menu selection approach based on finger counting [30] in which all the menu items are numbered and the user has to extend a corresponding number of fingers to select a given item. It was shown to be a viable option for 3D menu selection tasks. Nonetheless, items have to be numbered, and the fingers will be used for counting, and are thus dedicated to this operation and cannot be used for other different tasks. Table 2.1 summarizes the above explained limitations.

2.4 Three Fingers Clicking Gesture

This section explains in details the Three Fingers Clicking gesture. First, the difference between machine learning and heuristics regarding gesture detection is explained. The gesture detection algorithm is then explained, followed by an evaluation. Finally, the results and limitations of the gesture are discussed.

Table 2.1: Summary of existing work.

Reference	Limitation
[9]	Users hover their hand over an onscreen button, and have to wait for 1 second to trigger the action. This introduces waiting time and does not allow the user to cancel his operation after the trigger time has elapsed.
[11, 28]	Use a crossing technique; the crossing event occurs when the cursor intersected with the boundary of a graphical object. While effective, these crossing techniques are limited to a basic selection operation.
[10]	Uses an “index pointing, thumb up” gesture to perform a selection, which is not an intuitive selection gesture since it is not related to the operation it is associated with.
[29]	Uses postures which were defined by which fingers were raised or folded, and whether the thumb was stretched out, aligned with the palm, or tucked into the palm. While the vocabulary is increased, the system cannot be used for precise input.
[30]	Menu items are numbered and the user has to extend a corresponding number of fingers to select a given item. This forces the fingers to be dedicated to counting and hence cannot be used for other operations.

2.4.1 Machine Learning vs. Heuristics-based

3D gesture recognition methods can be mainly divided into two parts: machine learning and heuristics-based [31]. In the case of machine learning recognition, important features are extracted from the data and are then used as input for a classification algorithm. In addition, training is needed to make the classifier more robust and to maximize accuracy, as explained in 2.2.3. Conversely, in a heuristic based approach, no machine learning algorithms are used. For example, if users are playing a video game that they can control with movements of their body, they might need to perform a jump action, which is easily detectable using a heuristic algorithm: the head’s position is tracked, and a height threshold is defined. Whenever users jump, the algorithm checks if the difference of the head position during the jump and its position while standing is greater than the defined threshold, then a jump is detected.

2.4.2 Gesture Detection

While machine learning based recognition is more commonly used ([22]), we have opted for heuristics-based recognition. A simple heuristics-based solution for detecting an index-tap would be to check when the distance of the index finger to the camera is smaller than the distances of the other fingers. However, this approach is flawed in the fact that if the hand is not perfectly parallel to the depth camera, (i.e. if the hand is tilted in such a way where the little finger is the closest finger to the camera) the distance of the index to the camera will never be smaller than that of the middle, ring and little fingers'.

In the "Three Fingers Clicking Gesture", we introduce a new approach to detecting a midair tap gesture. A SoftKinetic DS325 depth sensing camera which uses Time-of-Flight technology [32] was used to detect the gesture. The DS325 has a 320 x 240 depth pixel count and 74° x 58° x 87° (Horizontal x Vertical x Diagonal) depth field of view. The nominal operation range is 0.15 m - 1.0 m. Using the DS325 Software Development Kit (SDK), the 3D position of the thumb (T), index (I), middle finger (M) and palm center (P) were retrieved. From those positions, the vectors $\overrightarrow{PT}$, $\overrightarrow{PI}$ and $\overrightarrow{PM}$ were created. Since a plane can be defined from 3 points, the plane [P, T, M] was defined from the respective 3D coordinates of the palm center, thumb and middle finger. For this reason, the thumb and middle finger had to always stay extended. This plane now represents the palm of the hand. The normal vector $\overrightarrow{PN}$ protruding from the palm is then computed as the cross product of vectors $\overrightarrow{PT}$ and $\overrightarrow{PM}$. Next the dot product of $\overrightarrow{PN}$ and $\overrightarrow{PI}$ is calculated, from which the cosine of the angle $\angle$ NPI can be retrieved. The value of $\angle$ NPI is then deduced by using the arc cosine function. The angle $\theta = 90° - \angle$ NPI is then computed; it represents the angle between the index finger and the palm of the hand. The tapping gesture is performed by executing a tap with the index finger. When the latter moves, θ varies, and when it crosses a given threshold (an angle of 12° was used in this study), a tap is detected. The DS325 SDK allows the detection of the 3D coordinates of the fingers and palm center, but cannot retrieve the palm normal vector stated above; we had to compute it manually and therefore the little and ring fingers had to be folded to prevent any modification of the plane [P, T, M] due to unintended movements of the middle fingers which can be caused by the moving of the little and ring fingers. Figure 2.1 shows the Three Fingers Clicking gesture, and the algorithm of the click detection is summarized in Algorithm 1. This method does not need a calibration step to be usable; users can start using it immediately. Another advantage is the fact that the reference (plane [P, T, M] and vector $\overrightarrow{PI}$) is always using with the hand, and thus the users could rotate their hand in any direction, and the tap would always be detected even if the distance of the index finger to the depth sensor was smaller than that of other fingers.

Algorithm 1 Three Fingers Clicking Algorithm

```
procedure OnFrame
    Get the 3D position of thumb (T), index (I), middle finger (M) and palm center (P)
    Define the plane [P, T, M]
    Deduce normal vector $\overrightarrow{PN}$ perpendicular to the palm
    Compute ∠ NPI from the dot product of $\overrightarrow{PN}$ and $\overrightarrow{PI}$
    Compute angle $\theta = 90° - ∠$ NPI
    if ($\theta >$ Threshold) then
        GenerateClick()
    end if
end procedure
```

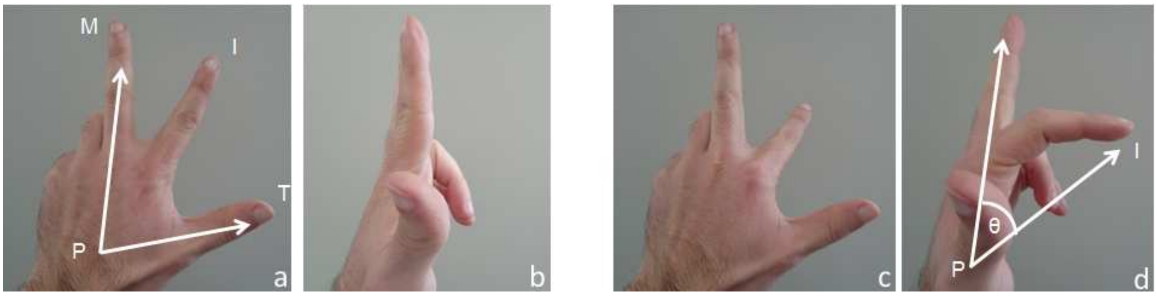


Figure 2.1: The Three Fingers Clicking gesture: a and b illustrate an unclicked state; c and d represent a clicked state.

2.4.3 Evaluation

A prototype was built using a SoftKinetic DS325 depth sensing camera and its SDK. The camera was placed on top of a 23” monitor with Full HD resolution, facing the user and tilted down approximately 10°. The algorithm was implemented on a computer equipped with an Intel Core i5 3.2 GHz CPU. The SDK of SoftKinetic was used to detect hand tip positions. Onscreen rendering was implemented in Allegro [33]. The entire prototype was written in C++. To evaluate our system, an experiment was conducted. 9 participants (5 males, 4 females) aged between 23 and 30 volunteered; 6 among them are computer scientists/engineers. 8 of them are highly familiar with computers. 2 of them are left-handed. They were instructed to hold open the thumb, index and middle fingers of their preferred hand an to fold the ring and little fingers. They were then asked to perform taps with their index finger until 20 taps were detected. The total number of taps was counted to spot false detections. The obtained success rate was 89.56%.

2.4.4 Discussion

Using 3D coordinates allowed the gesture to be detected regardless of the hand's position or rotation. It was also detected even if the hand was moving. In this experiment, the camera was facing the user. However, the Three Fingers Clicking gesture can also be detected even if the camera is behind the users' palm. This can be used in a tabletop setup, where the depth sensing camera is pointing downwards.

Since the gesture relies on the detection of three fingers, this can be a detection limitation. When a given hand is in its own half of the camera space, the three fingers were easily detected. However, when the hand moved into its opposite half of the camera space, the finger detection failed, even if the hand was still in the camera's field of view. This is due to the fact that when the hand crosses into the opposite space, the thumb and the index are occluded by the middle finger, and the camera fails to keep track of them (Figure 2.2). Another limitation is the gesture itself: 6 participants reported that keeping the three fingers held open stressed their forearm's muscles quickly, and found some difficulty in maintaining the gesture.

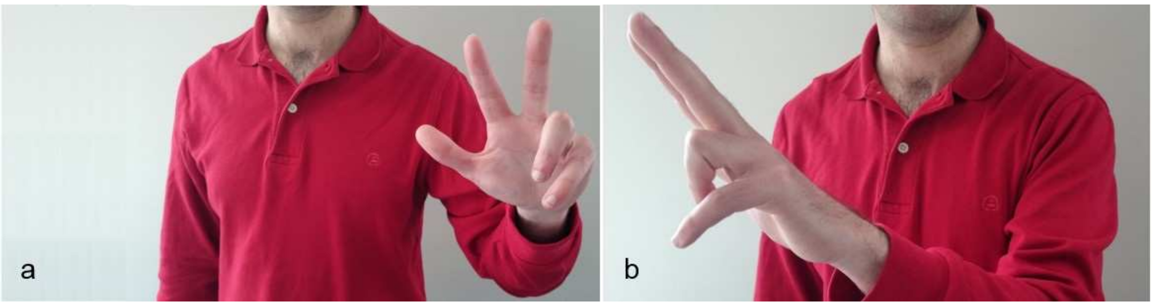


Figure 2.2: Limitations of the Three Fingers Clicking gesture: The fingers of the left hand in the left half of the camera space are easily detected (a), however, detection fails when the left hand moves to the right half of the camera space (b).

2.5 Depth Click

By using the Three Fingers Clicking gesture (2.4), a midair tap was successfully detected. However, since the thumb and middle fingers were always extended to generate a plane, they could not be used for other actions. Moreover, the ring and little fingers had to be always folded. Those conditions had 2 effects on the functionality and ergonomics of the gesture.

First, since the thumb, middle, ring and little fingers were dedicated to specific poses, they could not be used for tapping, and thus a tapping gesture was only limited to the index finger. Furthermore, maintaining the thumb and middle fingers extended and the ring and little fingers folded stressed the hand's muscles and as a result, the users' hands would get tired quickly. Maintaining the gesture proved difficult, which reduced its ergonomics.

To overcome the previous limitations, we introduced "Depth Click", an amelioration over the Three Fingers Clicking gesture. In this section, we explain about the improvement and detection of midair finger clicks, then we move on to the evaluation of Depth Click. We then discuss the approach. Finally, we explain in details the prototype application that uses Depth Click as means of interaction.

2.5.1 Gesture Detection

A Creative Senz3D depth sensing camera [34] using Time-of-Flight technology and capturing videos at 30 fps, was used for the gestures detection. The Intel Perceptual Computing SDK was utilized to retrieve the 3D coordinates of the Thumb (T), Index (I), Middle Finger (M), Ring Finger (R) and Little Finger (L), as well as the palm center (P). By using the coordinates of those points, the vectors $\overrightarrow{PT}$, $\overrightarrow{PI}$, $\overrightarrow{PM}$, $\overrightarrow{PR}$ and $\overrightarrow{PL}$ are computed (Figure 2.3 - a). The normal vector $\overrightarrow{N}$ perpendicular to the palm is also retrieved (Figure 2.3 - b). To detect a tap (Figure 2.3 - c, d) the angle α between $\overrightarrow{N}$ and any of the previously mentioned vectors is computed (using the same approach as in 2.4.2). Then the complement of this angle is computed ($\theta = 90° - \alpha$). θ represents a value that is used for a tap detection by comparing it to a threshold; if a finger crosses this threshold (an angle of 12° was used in this study), we suppose that this finger is now in a click position.

This approach led to a series of enhancements over the Three Fingers Clicking Gesture (2.4). First, users do not need to keep their ring and little fingers folded anymore; this proved to be easier to maintain than the previous one. Moreover, those two fingers can now be used for tap detection, and can thus have their own dedicated taps. Furthermore, the thumb and middle fingers are not needed anymore to generate the palm plane for tap detection, which also means that they can have their own dedicated taps detected independently. Finally, since the ring and little fingers do not need to be folded at all times, and since the thumb and middle fingers are not required to be extended, the neutral position (no click is occurring) became a more relaxed open palm gesture.

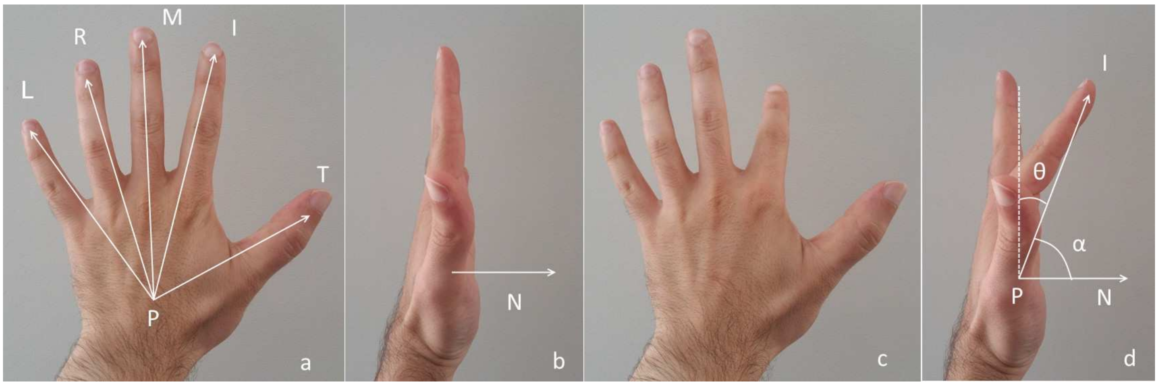


Figure 2.3: Depth Click: a and b depict the front and side views of an unclicked state. c and d show the front and side views of an Index-click gesture.

2.5.2 Click Down and Click Up

To make the tapping gesture more robust, another improvement was introduced in the way a it is detected. Simply θ crossing the threshold (2.5.1) is not enough anymore. In the Three Fingers Clicking Gesture (2.4), once the index finger crossed the threshold, a continuous tap event was being fired; this resulted in multiple taps in a row. To prevent this, we introduced a new rule, where a click gesture was now separated into two actions: a "Click Down" and a "Click Up". Table 2.2 summarizes those two gestures

Table 2.2: Click Down and Click up

Gesture	Explanation
Click Down	θ of a given finger crosses the threshold. This finger is now in Click Down mode
Click Up	A finger that was previously in Click Down mode retreats to behind its threshold. After this action, the given finger is now back into neutral position (no click is detected)

A click was therefore defined as a Click Down followed by a Click Up, for a given finger, as summarized in Algorithm 2. This approach now clearly defines the beginning and the end of a tapping gesture. To make the best use of the gestures, we implemented them using the Observer Design Pattern [35]. This design pattern is ideal for events and is typically used in GUIs. It dictates using two objects for an event: an Observable (which generates an event) and an Observer (which receives said event). Figures 2.4 and 2.5 show the Click Down and Click Up actions for the index and thumb respectively

Algorithm 2 Depth Click

```
procedure OnFrame
    Get palm center (P)
    Get normal vector $\overrightarrow{PN}$ perpendicular to the palm
    for every Finger F in hand do
        Get the 3D position of F
        Define the vector $\overrightarrow{PF}$
        Compute ∠ NPF from the dot product of $\overrightarrow{PN}$ and $\overrightarrow{PF}$
        Compute angle $\theta = 90° - \angle$ NPF
        if ($\theta >$ Threshold) then
            GenerateClickDown(F)
        else
            if (Finger has been in Click Down posture) then
                GenerateClickUp(F)
            end if
        end if
    end for
end procedure
```

2.5.3 Possible Gestures

Separating a tap into two different events not only prevented the generation of multiple tapping events in a row, but also introduced a series of improvements and possibilities to the interaction itself. A tap gesture is now limited by two boundaries (Click Down and Click Up), which makes its detection more robust. Furthermore, a click is not limited by a time constraint: users can keep their finger in Click Down mode as long as they wished, and a full click will only be detected after a Click Up. By using this method, a new "Long Click" event became possible. A long click was generated when users kept their finger in Click down posture for more than a given time threshold (500 ms was used in the prototype); once the click is released, the Long Click event is terminated. A Long Click and a regular click are detected separately, and hence can be used in the same interface independently.

Another gesture that we were able to detect is "Double Click", which is defined as two consecutive taps that occur within an interval of time that is less than a specified threshold (a 1000 millisecond threshold was used in the prototype).

All of the above mentioned gestures can be detected anywhere, as long as the hand was detectable by the depth sensor. However, they can also be used for precise input in the case of a user interface element. That is, that element cannot be activated unless the hand is hovering over it. The palm center of the hand was therefore used as a cursor, because

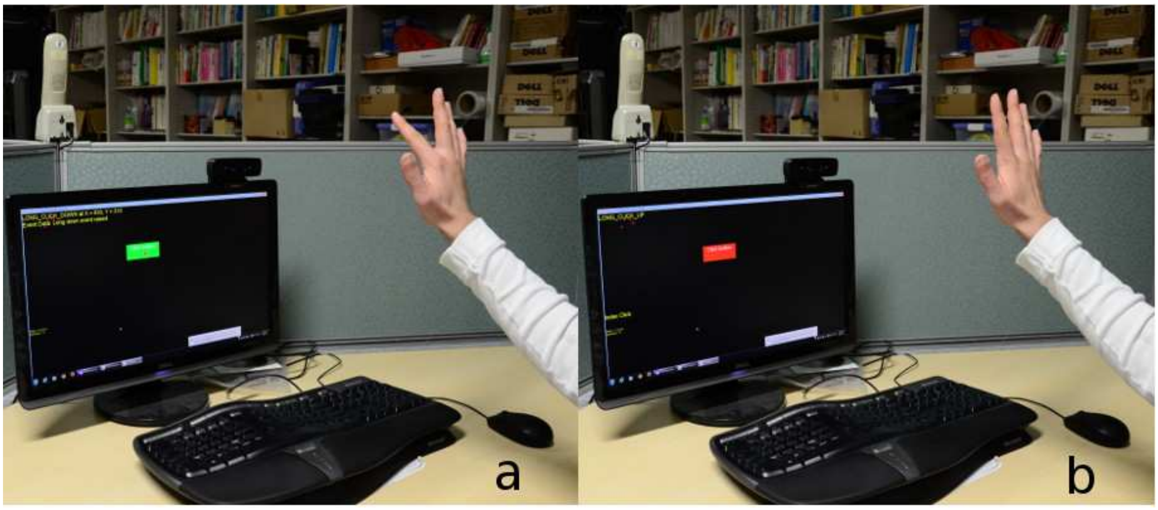


Figure 2.4: Index-click sequence: in a, the index finger executes a Click Down, and in b, it performs a Click Up. Note that the GUI element (button) changes color to green when it is clicked.

its location does not change when fingers are executing a tap (if the fingertip was used, its position will change when it is performing a tap gesture). As such, a user interface element can be clicked if the cursor intersects it. The separation into Click Down and Click Up have made the cancellation of the event possible. To accomplish this, a tap had therefore to satisfy an extra condition, in the case of a user interface. Not only it should it consist of a Click Down followed by a Click Up, but both those gestures have to occur inside the boundaries of the user interface element. Due to this, users can now cancel the event even if they started clicking an element. This can be used to reduce errors due to the activation of the wrong target. Because both Click Down and Click Up have to occur over the same target, users can cancel the action, even if they already started performing a Click Down, by simply moving the cursor outside of the element before releasing the click. By doing so, the Click Up event will not be detected over the interface element, and as a result, the click itself will be canceled and an accidental click will be prevented.

2.5.4 Evaluation

A prototype was built using a Creative Senz3D depth sensing camera which uses Time-of-Flight technology, capturing videos at 30 fps. The resolutions are 640x480 for RGB and 320x240 for depth. The camera was placed on top of a 23” monitor with Full HD resolution,

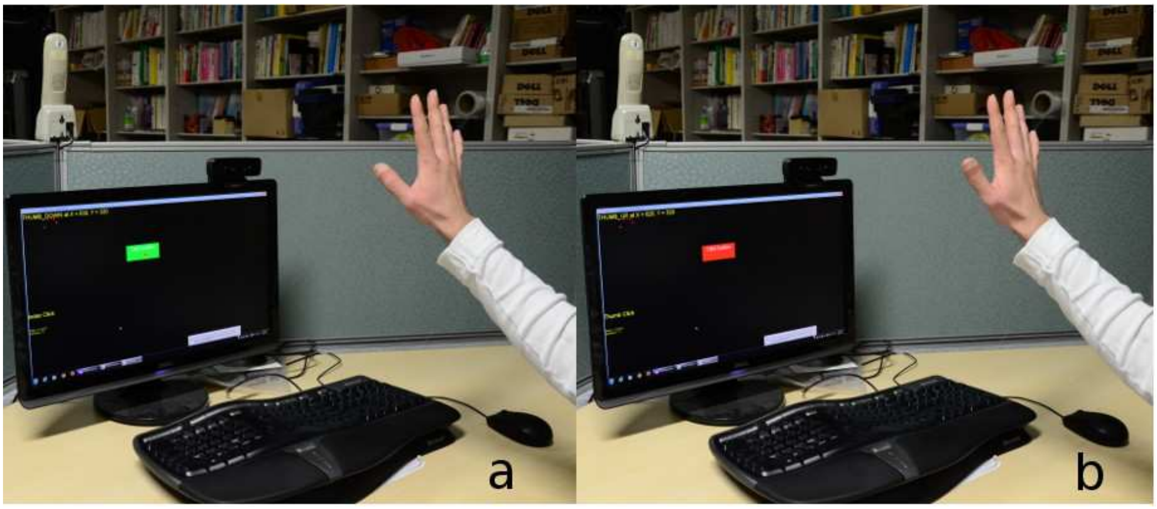


Figure 2.5: Thumb-click sequence: in a, the thumb finger executes a Click Down, and in b, it performs a Click Up. Note that the GUI element (button) changes color to green when it is clicked.

facing the user and tilted down approximately 10°. The algorithm was implemented on a computer equipped with an Intel Core i5 3.2 GHz CPU. The Intel Perceptual Computing SDK 2013 was used to retrieve the 3D positions of the fingers and palm center, as well as the normal vector protruding from the palm. Those points were used to construct the various vectors described in 2.5.1. Onscreen rendering was implemented in SFML [36]. The entire prototype was written in C++. To evaluate our system, two experiments were conducted and a questionnaire was filled in. 10 participants (5 males, 5 females) aged between 20 and 36 volunteered; 5 among them are computer scientists/engineers. All of them are highly familiar with computers. 1 of them is left-handed. After explaining the process of the experiments to the participants, they were given 2 minutes to practice the different interactions (Index-click, Index- Double-click, Index-long-click and Thumb-click). They were also given the chance to test the Image Gallery Prototype (2.5.10).

2.5.5 Experiment 1 - Comparison With Other Types Of Selection Mechanisms

In this experiment, Depth Click was compared with 3 other gestural selection mechanisms. 9 blue buttons, sized 100 x 100 pixels, were arranged in 3 rows of 3 buttons each, separated by 100 pixels horizontally and vertically (Figure 2.6). The buttons were numbered from 1 to 9. An additional Start button was also added. After selecting the Start button, a number is displayed at the top of the screen, and the participants had to select the corresponding button. Selecting a button with a different number than the one that was shown counted as an error. 20 selections were required. The number of errors and the completion time were saved. The participants were required to repeat this experiment 4 times, once per selection mechanism (80 selections per participant, for a total of 800 selections).

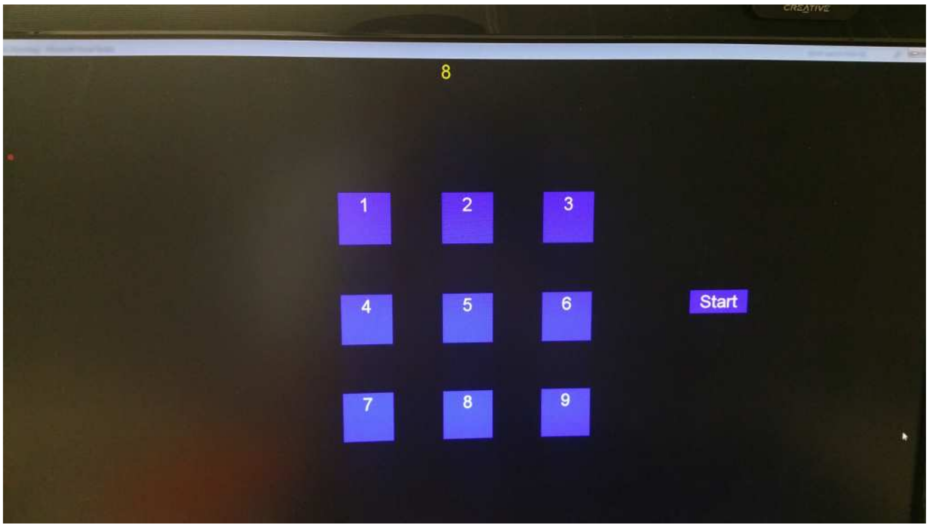


Figure 2.6: 9 buttons, plus the Start button at the right. A number at the top of the screen showed the button that should be selected.

A cursor showed the position of the palm center. Whenever the cursor hovered over a button, that button's color changed to red to indicate hovering to the user, and that button's color flashed to green upon a positive selection. The different selection mechanisms used in this experiment were:

- Depth Click (Click Down, then Click Up over the same button).
- Double Cross: the cursor has to hover over a button, exit the button's boundaries, then hover over the same button again to trigger a selection. Crossing an unintended target only once does not select it, and hence was not counted as an error.
- Cross: the cursor hovers over a button to trigger a selection. An unintended cross was counted as an error.
- Closed Hand: the user hovers over a button, then closes his hand to select it.

2.5.6 Experiment 2 - Detection Test

The participants were asked to perform 20 clicks over a single button sized 200 x 100 pixels. They were required to repeat this experiment 3 times (60 clicks per participant, for a total of 600 clicks), each time using a different click (index-click, thumb-click and index-double-click). The number of detections per click type were then checked.

2.5.7 Questionnaire

After completing the experiments, the participants were asked to fill the following questionnaire, to which they could respond on a five point Likert scale (-2 = Strongly Disagree, 2 = Strongly Agree):

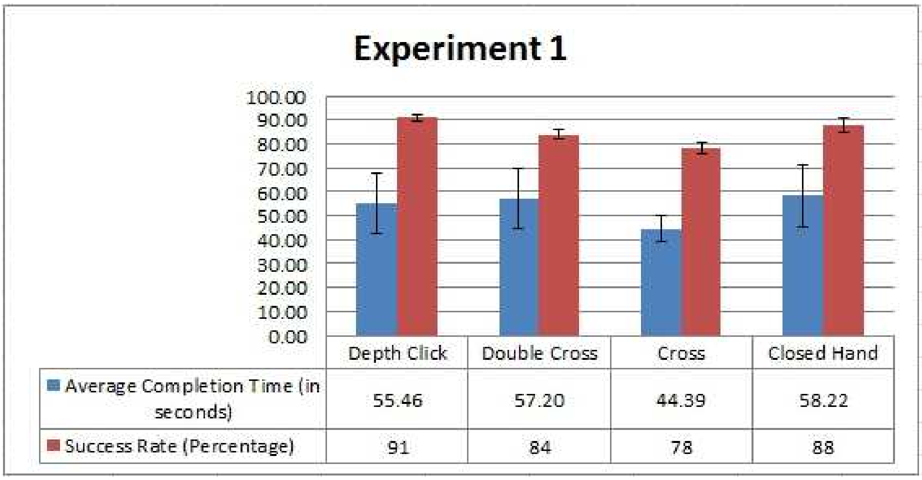
1. Is an Index-click easy to perform?
2. Is a Thumb-click easy to perform?
3. Is a Double-click easy to perform?
4. Does our method feel more natural than other selection mechanisms?
5. Does a Long Click feel more natural than a Closed Hand for performing dragging actions?

In the last question (What is your preferred selection mechanism?) the participants had to choose an answer out of the following 4 possibilities: Cross, Double Cross, Click, Closed Hand. In addition, the participants were offered the opportunity to enter their reason, and overall comments, freely.

2.5.8 Results

In the first experiment, the mean time to perform 20 selections using Depth Click is 55.45 seconds (S.D. = 12.58). While it is slightly faster than Double Cross (mean 57.2, S.D. 12.61): T(18) = -0.3, p = 0.76, the difference is not significant. Depth Click was slower than Cross (mean 44.4, S.D. 5.54): T(12) = 2.54 , $p < 0.05$, showing that the difference is considerable. Finally, Depth Click is also slightly faster than Closed Hand (mean 58.21, S.D. 12.61): T(18) = -0.48, p = 0.63, but the difference is not significant. It is to be noted that the times include the processing time that was performed to detect the various gestures. Depth Click had a success rate of 18.2 (91%) (S.D = 1.39), higher than Double Cross (mean 16.8 (84%), S.D. 1.87): T(17) = 1.89, p = 0.07. It also had a success rate higher than Cross (mean 15.6 (78%), S.D. 2.36): T(15) = 2.99, $p < 0.05$, and higher than Closed Hand (mean

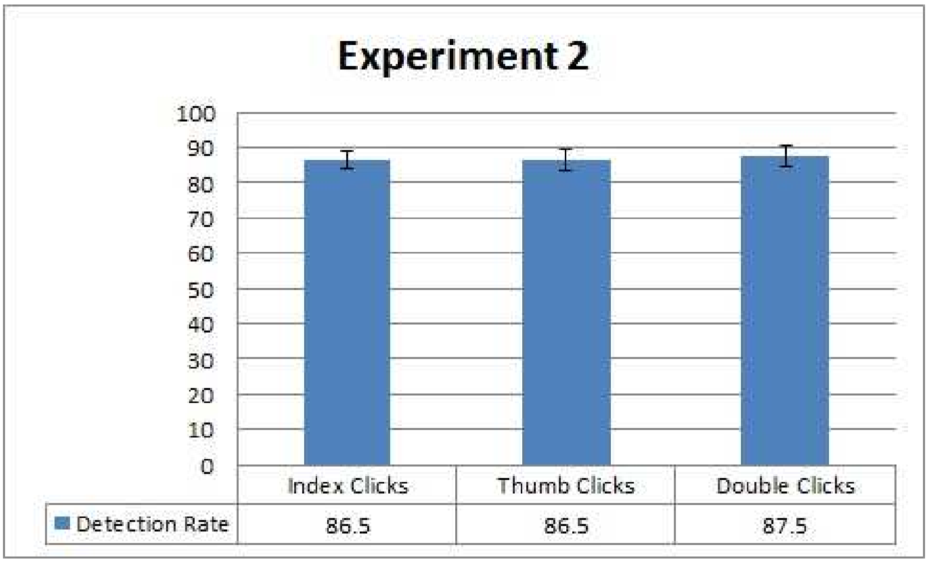
17.6 (88%), S.D. 2.95): T(13) = 0.58, p = 0.57. The results of Experiment 1 are shown in Figure 2.7



	Depth Click	Double Cross	Cross	Closed Hand
🟦 Average Completion Time (in seconds)	55.46	57.20	44.39	58.22
🟥 Success Rate (Percentage)	91	84	78	88

Figure 2.7: The results of Experiment 1.

In the second experiment, out of 20 clicks, the average numbers of detected Index-clicks, Thumb-clicks and Double-clicks are respectively 17.3 (86.5%, S.D. 2.21), 17.3 (86.5%, S.D. 3.02) and 17.5 (87.5%, S.D. 3.2). Figure 2.8 shows the results of Experiment 2.



	Index Clicks	Thumb Clicks	Double Clicks
🟦 Detection Rate	86.5	86.5	87.5

Figure 2.8: The results of Experiment 2.

As for the questionnaire, all the averages were above neutral. Most people agreed that an Index-click is easy to perform (mean 1.2, S.D 0.63) and many agreed that a Thumb-click is easy to execute (mean 0.6, S.D 0.84), same for a Double-click (mean 0.9, S.D 0.73). Most participants strongly agreed that a click is more natural than the other proposed selection mechanisms (mean 1.4, S.D 0.69), and many felt that a Long-click is more natural for dragging (mean 0.7, S.D 1.15). 8 out of 10 participants (80%) chose clicking as their preferred selection mechanism.

2.5.9 Discussion

Depth Click was faster than Double Cross and Closed Hands; however, the difference in completion time was not significant. It also had a higher success rate than those methods, but nonetheless, the difference was not considerable. And while it was slower than Cross, Depth Click's error rate was significantly lower than that of Cross. And with detection rates higher than 86%, we can say that the method is robust and reliable. The reason why Cross is fast is that the cursor only needs to intersect the button to trigger the selection; however, this makes it more error prone as it is easy to hover over unintended buttons. Depth Click requires a tapping motion; the gesture itself and the processing time to detect it result in an interaction slower than Cross. However, Depth Click has higher reliability regarding errors in selections.

As per the answers of the questionnaire, it can be deduced that tapping is more natural than the other selection mechanisms, and it was the preferred one.

One of the problems that we faced was the fingers detection. Since we were using the camera's SDK to retrieve the fingers' positions, the detection of a tap was constrained by the camera's abilities to detect the fingers. The camera proved to be sensitive to strong ambient light, and in many cases, the hand would stop being detected in many frames, even if the user did not move said hand. Since the algorithm relies on having a Click Down and Click Up being performed over the target, if the palm center's detection fails during the Click Up gesture, it would be considered as a canceled click, and thus not detected. The detection problem especially affected the Long-click: while being pressed down, the finger might not be detected and thus the interaction would end.

Depth Click's design gave it some advantages over other techniques that could not be quantified in experiments. Those possibilities are listed in Table 2.3.

Table 2.3: Depth Click's possibilities compared with other techniques.

Possible Action	Depth Click	Double Cross	Cross	Closed Hand	Hover
Detection not limited by time	Yes	Yes	Yes	Yes	No
Allows a change of mind	Yes	Yes	No	No	Partial
Allows a "long press"	Yes	No	No	Yes	No
Unaffected by the interface layout	Yes	Partial	No	Yes	Partial
Can be used for actions other than selections	Yes	No	No	No	No
Detectable without a target	Yes	No	No	Yes	No

2.5.10 Image Gallery Prototype

An image viewer prototype with functionalities similar to that of a smartphone's gallery was implemented, and control was provided with Depth Click. Three possible interactions with the displayed images were defined: "Swipe", "Zoom" and "Rotate". Figure 2.9 shows the zooming and rotating functionalities in action.

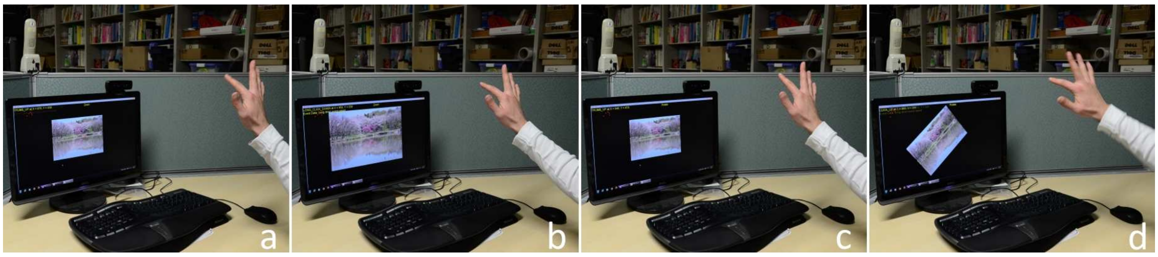


Figure 2.9: Users click (a) and push their hand closer to the camera to zoom in (b); to rotate, they click (c) and turn their hand (d).

Swiping allows users to change photos. In "Swipe" mode, the action is triggered using a long Index-click-down. The cursor (determined from the palm center as explained in 2.5.3) has to be above the photo, otherwise it cannot be swiped. Once a Long-index-click-down is detected, the position of the cursor is then saved. The users can then move their hands left or right, and the photo will move along. When an Index-click-up is detected, the position of the cursor is retrieved, and the horizontal distance between the Click Down position and Click Up position is measured. If the distance is greater than 300 pixels, the previous/next image is displayed.

Zooming gives the users the opportunity to zoom in/out of photos. In "Zoom" mode, just like in "Swipe" mode, the action is triggered using a Long-index-click-down. The cursor has to be above the photo, otherwise the zooming action cannot be accomplished. Once a Long-index-click-down is detected, the distance of the index finger to the depth sensor (Z coordinate) is saved. At this moment, users can either push their hand closer to the depth sensor to zoom in, or away from it to zoom out, all while the index is in Click Down position. When the click is released, the distance of the index to the depth sensor is again measured. If the new distance (on Click Up) is smaller than the old one (on click-down) then the finger got closer to the sensor, and a zoom-in action is executed. On the other hand, if it is greater, then the finger got further from the sensor, and a zoom-out action is performed. A 0.1 difference in the scaling factor was added or removed in every frame to implement the zoom-in and zoom-out effects respectively. The overall scaling factor was limited between 1.0 and 2.0 to prevent users from performing extreme zooming.

Rotating allows users to rotate photos. In “Rotate” mode, the action is triggered using a Long-index-click-down. The cursor has to be above the photo, otherwise it cannot be rotated. Once a Long-index-click-down is detected, the positions of the thumb (T) and index (I) are saved, and the vector $\overrightarrow{TI}$ is created. Now the users can rotate their hands clockwise and counter-clockwise, and the photo will rotate along. This is accomplished by continuously retrieving the vector $\overrightarrow{TI}$ and measuring the angle between it and the original $\overrightarrow{TI}$. The photo is then rotated by this angle. A Click Up stops the rotation interaction.

Since those three interactions are executed in midair, it is possible to unintentionally activate two or all of them at the same time. For example, when users are performing a Swipe, if their hand moves closer to the sensor, then Zoom will be also executed. To prevent this from happening, we included a cycling mechanism; this forces users to only use one interaction at a time. Furthermore, they need to cycle between the available options in order to use them. To cycle between those three functionalities, users would execute a Thumb-click anywhere in the screen. Every tap of the thumb would then change the interaction mode to one of the above cited three functionalities. When the applications starts, it is in Swipe mode. A Thumb-Click would put it in Zoom mode, and another one would activate Rotate mode. The next Thumb-click would reactivate Swipe mode, and so on.

2.6 Summary

In this chapter, we presented methods on finger tap detection using a depth camera. In the first approach, we used the thumb and middle fingers to generate a plane. which was employed to determine the angle of the index finger in regards to this plane. The angle was compared with a threshold to determine whether an index tap was executed. It had good performance, but was tiring to use and limited to the index finger. In the second approach, we were able to detect the angle of every finger in regards to the hand's palm, thus any finger could be used to perform taps. Furthermore, the tap gesture was separated into Click Down and Click Up gestures, which increased the possibilities for the usage of taps and gave the users control over the start and stop timing of the gesture. We were able to detect additional gestures such as Long Click and Double Click. Moreover, in the case of a user interface, we added an extra rule to detect a click, which stipulated that both Click Down and Click Up gestures should be executed over the interface element, which reduced unintended selection errors and allowed users to change their mind even after starting a click operation. Finally, we implemented an image viewer prototype which uses clicks as a mean of interaction, and the clicks were used to start more complicated gestures that could be executed in 3D. This approach still had limitations such as a tiring neutral position and the lack of use of precision of the taps, which formed the motivations for the next step of our research that is described in details in Chapter 3.

Chapter 3

An Algorithm For Explicit Midair Finger Clicks Detection

In this chapter, we describe in details Xpli Tap, an algorithm to explicitly detect midair finger clicks, as well as its midair keyboard application. The design of the system is described. First we describe the movements of the human fingers, then we detail the algorithm steps that we used to reliably detect midair taps. We later explain how a midair keyboard was implemented and the assumptions behind it, including three different approaches for detecting a keyboard row. We then conduct experiments to determine the reliability of the approaches. Finally, we discuss the capabilities and limitations of the system and describe some possible applications.

3.1 Research Goals

Previously, we have proposed two selection mechanism algorithms for finger tap detection using a depth camera. However, we have found that they had some limitations. The Three Fingers Clicking Gesture (2.4) was limited to a simple index tap because the other fingers were not usable, and it was difficult to maintain. Depth Click (2.5) improved on that, but still suffered from limitations of its own. While it could detect the taps from all the fingers, it was not precise: the position of the fingers and their taps was not usable. This limited Depth Click to a generic click, and using precise information from a fingers position was impossible. Moreover, the neutral position was an open palm, where the users had to keep their fingers fully extended, which caused tiredness after some use, since relaxing a finger might cause it to cross its threshold and might as a result generate an unintended tap. To overcome these limitations, we decided to make the identification of a tap more precise. To accomplish this, we will carefully identify the tapping finger. To further increase the precision, we want to detect the exact location of the tap when it occurs. Furthermore, we would like to make the gestures more ergonomic, by making them easier to perform, and customizing them for every user; this will also increase their accuracy.

3.2 Midair Tap

Using hand gestures for computer control opens up the possibilities for different kind of interactions, such as waving [9], “index pointing thumb up” gesture [10], checking the folded and extended fingers [29] and pinching [37]. Different gestures have different usage, and an important aspect of an interface usability is being able to select its different components. We have covered using a finger tap gesture as a selection mechanism in Chapter 2. However, a midair finger tapping gesture can extend beyond that, especially if additional features of the fingers and gesture can be taken advantage of. In 3.2.1, we first start by discussing the importance of a midair tap.

3.2.1 Importance Of Midair Tapping

Van de Camp et al. [27] investigated the interaction with distant interfaces while using gestures, which they separated into three groups, depending on the body part that was used for those gestures: the fingers, the hand and the arm. They performed their study on the following gestures: push, pull, rotate, point, dwell, bend, air-tap, pistol and grab. Since every gesture’s accuracy depends on its respective recognition system, they used a Wizard of Oz setup [38] where the participants interacted with a pretense system that was actually controlled by a hidden human experimenter, which allowed the participants to experience each gesture as if its recognition system was working perfectly. The participants chose air-tapping as their favorite input gestures, and many of them pointed out that it is very intuitive because it has a high resemblance to the use of a computer mouse. While air-tapping in the above study was performed with the index finger while having all the other fingers folded, in our study we used all the fingers to make the most use of the gesture.

3.2.2 Related Work

In uTrack [39], two magnetic sensors were worn on a user's fingers, and the sensors' readings were analyzed thus enabling the use of a combination of two fingers as an input device. The two sensors were attached to a finger different from the thumb (e.g. the ring finger), and a permanent magnet was attached to the thumb. The second sensor removed the ambiguity in locating the magnet's position, which in turn was used to control a cursor on a computer screen. This solution requires the attachment of sensors to the hand and is limited to only one finger.

Lin et al. [40] created an imaginary air touch panel that can be anchored in midair at any location that the user wills. After positioning the panel, users can interact with it by executing a quick tap on it.

In [41], Bai et al. use the fingertip to create an Augmented Reality (AR) interaction system for mobile devices. To utilize with the system, a finger is put behind the screen so that it can be detected by the device's camera; keeping the finger still in a small area for a defined amount of time changes the state of that finger so that it can interact with the device.

Similarly, Hürst et al. used finger tracking for AR interaction with a mobile device [42]. They used a marker on the fingertip they wished to track, and that finger had to hover an object from the real world until the associated progress bar filled indicated that the object has been selected. Baldauf et al. [43] implemented a markerless fingertip detection that was used to create natural interactions with mobile devices; several interactions became possible: a fingertip can be used to select a virtual object by pointing at it, and a pinch gesture with the index and the thumb can be used to grab a virtual object or to control an imaginary volume knob by turning it. Micro input devices systems (MIDS) utilizing microelectromechanical systems were used in [44] to create wearable devices that can function as a computer mouse or keyboard. A MIDS ring was made using accelerometers and was used to detect a finger tap motion; it was shown that it could handle the operations of a keyboard or mouse.

A flood filling technique was used in [45] to detect a tap by touching a surface; here, a depth map retrieved from a depth sensor was analyzed to determine if a finger intersects with a solid object. When a finger comes in touch with a surface, the flood filled area's pixels are counted, and when they cross a determined threshold, that finger is deemed to be tapping.

A finger clicking interface for Augmented Reality systems was presented in [46], since it feels more natural for a user to interact with a virtual object in a similar way he uses his hands to interact with objects in the real world. The limitation of this approach is that the click is used just as a selection mechanism, and the only finger used for clicking is the index finger. All of these works either attach some sensor to the users' hands and fingers, force the users to touch a physical object for tap detection or have a limited scope of possible interactions such as being able to use just one finger or detecting a simple tap that cannot be used beyond a selection mechanism purpose.

3.3 Human Fingers Motion

Since we are detecting midair finger taps, understanding the workings of the human fingers and the anatomy of the human hand is crucial. By understanding the behavior of the fingers, we can then create better algorithms that take their motions into consideration to get a higher accuracy.

According to Zatsiorsky et al. [47], "force enslaving" is the phenomenon under which fingers produce forces, even though subjects were using other fingers: even when the subjects were required to perform pressing gestures with one, two or three fingers, without lifting the other fingers, those other fingers still generated some forces. Li et al. [48] studied the motion of a given finger (master finger) and its effect on neighboring fingers (slave fingers), by measuring the angular displacements of the fingers. They found out that slave fingers tend to move when the master finger moved. They call this phenomenon "motion enslaving". They suggest that the enslaving effect was mostly observable on neighboring fingers, and that the middle and ring fingers have an enslaving amount of more than 60% of their own maximum angular displacements. They also suggest that the index finger had the highest level of independence. In [49], Häger-Ross et al. state that normal human subjects produced motion in other fingers, even when instructed to move only one digit without moving the others. They also suggest that the independence level between the fingers did not depend on the subjects' handedness, as they could not find a difference in finger movement independence between the dominant and non-dominant hands.

As can be seen in the above mentioned studies, human fingers move together, even if subjects are asked to move just one. This creates a particular challenge in midair tap detection: since there is no physical object to prevent the noninstructed fingers from moving, multiple taps from different fingers will be detected simultaneously due to the unintended motions created in the slave fingers. This can be problematic in the case of precise input, or if the input depends on the fingers (i.e. if different commands are assigned to different fingers).

3.4 Xpli Tap

Xpli Tap is the algorithm that we developed to explicitly detect midair finger taps. We use Depth Click (2.5) that we have previously discussed in Chapter 2 as our starting point.

Depth Click was shown to be a fitting selection mechanism, and less prone to generating errors (mis-selections) than existing gestural selection mechanisms, especially in a situation involving many targets packed in a tight area. Results have shown that Depth Click's success rate in choosing a specific target among many was higher than other selection gestures.

However, Depth Click still had some limitations. One of them is the fact that individual finger taps could not be distinguished: for example, if both the index and ring fingers crossed the threshold at the same time, the system would simultaneously register and Index Click and a Ring Finger Click, and would thus prevent the users from individually assigning functionalities to every finger. Another limitation is the neutral position of the hand. When the users did not want to tap, they had to keep their fingers extended. This caused tiredness in the hand muscles, since any bending of the fingers would create an angle greater than the threshold and as a result generates an unintended tap. Moreover, Depth Click detected a "generic click", that is, once the threshold was crossed, a click was detected regardless of the position of the finger, and as such, during a clicking operation, using precise information from a finger's position is impossible.

To overcome these limitations, we first introduced a "Relaxed Neutral Position" where the users could keep their fingers relaxed even when they are not executing any tap. Furthermore, by carefully examining the movements of the human fingers (3.3) we wrote an algorithm that isolates the tapping finger. We then proceed to the actual detection of the tap gesture. The above mentioned steps will be explained in details in the following sections.

3.4.1 Relaxed Neutral Position

“Neutral Position” is the term we use to indicate a hand posture during which no tap occurs. In Depth Click (2.5) a tap was detected once a finger crossed a given threshold. This threshold was the same for all users; this worked well on some individuals, but not on others. Moreover, the threshold was the same for all fingers, which forced the users to keep their fingers fully extended when in the neutral position. This created tension in the hand and forearms muscle after repetitive use.

Our goal is now the creation of a “Relaxed Neutral Position” to overcome the above mentioned limitations. In this new posture, we suppose that when no tap action is being performed, the fingers should be in a relaxed position that is ergonomic to the users, rather than fully extended as the previous one was defined. Figure 3.1 illustrates the difference between the original neutral position (Figure 3.1 - a) and the relaxed neutral position (Figure 3.1 - b).

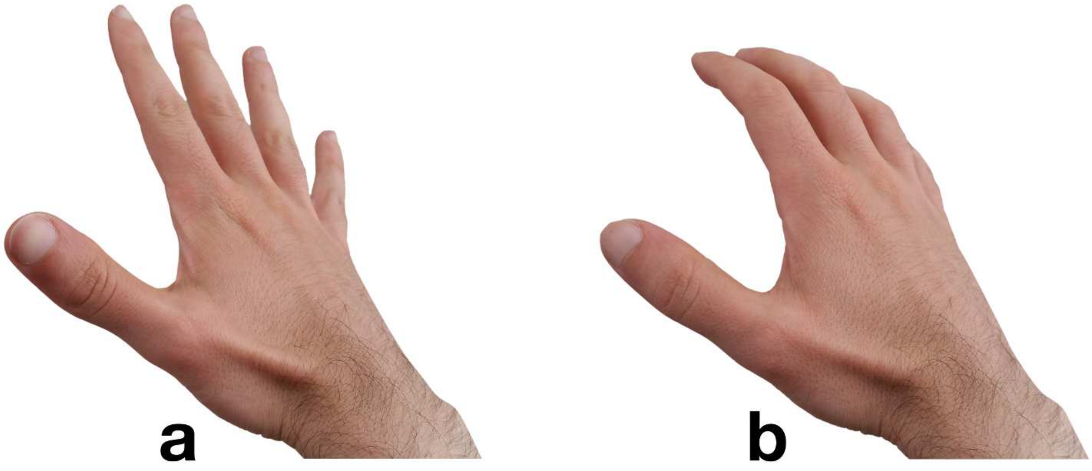


Figure 3.1: Difference in neutral postures of Depth Click (a) where the neutral posture assumes that the fingers should be fully extended when not clicking, and Xpli Tap (b) in which the fingers can be relaxed.

To accomplish the new position, and make it as ergonomic as possible, we had to take into consideration the following two points:

- Every finger of the same hand has a threshold angle θ that is different from the other thresholds of the remaining fingers of that hand.

- Even for the same finger, different users have different thresholds, and varied ways of holding the relaxed posture of that given finger.

This has lead us to implement a calibration mechanism that allows the creation of thresholds for each finger separately. Furthermore, those thresholds should be customizable for each and every user according to his/her preferred neutral position. We explain this calibration system in 3.4.2.

3.4.2 Calibration

Before describing the details of the calibration mechanism, we first start by explaining the coordinates systems that are used by the Leap Motion sensor.

The Leap Motion controller uses a right-hand coordinates system as can be seen in Figure 3.2 - a. According to [50], x, y and z coordinates of a finger tip are expressed in millimeters as the distance from the controller itself. That is, the frame of reference is the Leap Motion sensor itself, which in this case represents the origin 0, 0, 0.

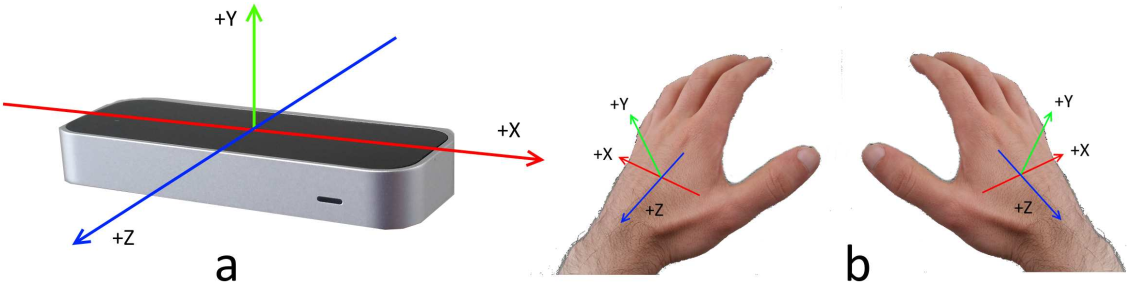


Figure 3.2: Absolute (a) vs. Relative (b) coordinates systems.

This “world coordinates system” is useful when measuring the position of the hand or fingers in respect to the sensor, or when checking the distance traveled by the hand. However, in some sections of our work such as when the exact reading of the position of the finger during its tapping motion is required, we would like to measure the movements of fingers according to the hand itself rather than the controller. The Leap Motion SDK provides a functionality to allow us to transform world coordinates (absolute coordinates) to ”local coordinates” (relative coordinates) where they become in the hand’s frame of reference [51]. Here, the palm position becomes the origin, therefore the coordinates of the fingers are retrieved according to this new origin, rather than the controller. The relative coordinates system is shown in Figure 3.2 - b.

To start the calibration process, the users put their hands above the Leap Motion controller, while our system is reading the required information. This is performed during 200 frames, and takes approximately 2 seconds (software using Leap Motion runs at approximately 100 fps, but the frame rate varies depending on the available resources). During those 200 frames, we save the following information:

- The 3D coordinates of every fingertip, in world coordinates.
- The 3D coordinates of the palm center, in world coordinates.
- The 3D coordinates of every fingertip, in local coordinates.
- The 3D coordinates of the palm center, in local coordinates.
- The θ angle (Figure 2.3 - d) of every finger.
- The finger bones angle (explained in 3.5.3) of every finger.

After retrieving the above information, we compute the mean values for all of them. From the above values, we then compute the mean values of the θ angles. We call those mean values $\theta_{neutral}$. We then define the new thresholds, *for every finger*, as being $\theta_{neutral}$ *+ 10°*. We also calculate the mean values of the *Finger Angles* ∠ LPR, ∠ RPM, and ∠ MPI (Figure 2.3 - a). We call the mean values of these angles $\angle\ LPR_{neutral}$, $\angle\ RPM_{neutral}$ and $\angle\ MPI_{neutral}$ respectively. Those angles are assumed to be in the same horizontal plane as the palm; that is, the fingers' Y coordinates are projected to the palm. This is introduced so that a vertical movement does not affect the angle's value when the finger moves down and up, but rather only a horizontal movement does. We also compute the mean value of the distance between each fingertip and the palm center. We suppose that this is the neutral palm distance, that is, the distance between the fingertip and the palm center when the fingers are in their neutral (relaxed) position.

3.4.3 Elementary Tap Detection

As stated in 3.4, Xpli Tap is more than a simple tap detection algorithm. The first step in the algorithm is, however, an elementary tap detection mechanism. Once a tap can be detected, further improvements will be applied to make the system more robust and reliable. To accomplish that, we used Depth Click (2.5.1) as our starting point. We have also use the same Click Down and Click Up (2.5.2) approach that we had used in Depth Click: whenever a finger crosses its threshold, a Click Down event is fired; whenever that finger goes back behind its defined limit, a Click Up event will be raised. However, in Xpli

Tap, each θ threshold is going to be custom-defined for every finger, as well as for every user, as described in 3.4.2. An elementary tap can now be detected for every finger of the hand.

3.4.4 Determining The Tap Position

Detecting the tap gesture is quite useful for creating natural user interfaces. A possible way of increasing the gestures vocabulary is by determining the finger that is associated with the tap. This has already been accomplished in Depth Click, and is a feature of Xpli Tap (3.4.3). To further increase the vocabulary, it is necessary to use other features from the information available about the fingers. Such a feature is finding the exact position of the tap for each finger. Previously, we have used the palm center as a cursor (2.5.3, 2.5.10). This is quite advantageous when using 2D interfaces, and a cursor has to displayed onscreen to help guide the users, for example, to tap a button. In 3D applications where a cursor is not needed, other approaches can prove more suitable; such an approach is detecting the tap position for the tapping finger. This can help augment the gestures vocabulary; for example, we can calculate the distance between the palm center and the tap position. This distance can then be used when a tap is detected to distinguish new gestures. For example, if the distance is greater than a predetermined value, the gesture will be different than if the distance is smaller than that value. In this way, we have now, for each finger, two distinguishable taps, and thus the vocabulary has been augmented twofolds.

Since a tapping motion consists of a finger going in a down motion, followed by a going up movement, we have defined the “Tap Position” as the position of the fingertip (3D coordinates) when it reaches its deepest position during a Click Down (right before it starts going up). To achieve this, we supposed that the finger will be in its deepest position when its θ angle is at its greatest value. To determine when the angle of a clicking finger has reached its biggest value, we save the value of this angle in each frame after a finger has crossed its threshold. We also save the position of the fingertip. From the fingertip’s position, we deduce its distance to the palm center. We then proceed to subtracting the neutral palm distance (determined in 3.4.2) from the current distance. We then compare the values of θ across the frames: when a finger starts moving back up, its angle will start decreasing. We thus determine the value of the greatest angle for the clicking finger, as well as its associated fingertip position, distance from the palm center, and difference between this distance and the neutral palm distance. Algorithm 3 summarizes this step.

Algorithm 3 Determining the tap position

```
procedure OnFrame
    for every Finger in hand do
        if (AngleOfFinger > threshold) then
            Save $\theta$
            Save fingertip position
            Deduce PalmDistance = Distance(palm center, fingertip)
            Compute DistanceDifference = PalmDistance - PalmDistanceNeutral
        end if
        Determine the greatest $\theta$
        Determine its associated position and distances
    end for
end procedure
```

3.4.5 Tapping Finger Isolation

As explained in 3.3, human fingers tend to move together, even when we intend to only move one. Häger-Ross et al. [49] have found that noninstructed fingers generate a force when the instructed finger moves; the latter is however greater than that of the noninstructed fingers. The unintentional motion of these fingers does not affect the result of typing on physical keyboard, because their generated force is not enough to push a physical key. However, in the case of midair typing, there is no physical object to prevent an unintended tap from being detected.

To overcome this problem, we have implemented a “finger isolation algorithm”. Another point that we have previously explained in 3.3 is that according to Li et al. [48] the uninstructed fingers will move at a 60% angular displacement of the instructed finger. Based on this information, we supposed that during a tapping motion, the intended finger will move more than the unintended ones. To be able to find the finger that the user intended to tap with, we first defined the *Y-Travel* variable, which represents the difference of the Y-coordinates of a finger, between its clicking position, and its neutral relaxed position. That is, it represents the amplitude that is traveled by the moving fingers across the Y-axis. We therefore assumed that the finger which the user intends to use for a tap will have the greatest Y-Travel value during a tapping operation. Figure 3.3 - b explains how multiple fingers might cross the thresholds. The blue arrow illustrates the amplitude of the ring finger, whereas the yellow represents that of the little finger. Here, we assume that the latter is the tapping finger, and thus its Y-travel is the greatest between the two.

To determine this value, once one or multiple fingers cross their respective thresholds, we start saving their current position. We repeat this for every frame where at least one finger is crossing the threshold (that is, during every frame of a *Click Down* event). After saving the Y-travel value of every finger, we then proceed to computing the means of those values that have been saved across the previous frames until the current one. We then proceed to determining the greatest of the mean values. This operation is also performed in every frame of a *Click Down* event. This aforementioned result will represent the biggest Y-travel of all the tapping fingers; we then retrieve the finger that is associated with it. This will represent the intended tapping finger. We repeat this as long as one or many fingers are in clicking posture, as shown in algorithm 4.

Algorithm 4 Determining the tapping finger

```
procedure OnFrame
    for every Finger in hand do
        if (AngleOfFinger > threshold) then
            for every finger in hand do
                Find Y-Travel
            end for
            Determine the greatest Y-Travel
            CurrentFinger = finger with greatest Y-Travel
        end if
    end for
end procedure
```

3.4.6 Tap Detection

In Xpli Tap, detecting a tap requires multiple steps; in each of them different values and features are determined. So far, the following procedures are executed:

- Determining the relaxed position: a calibration mechanism is used here to allow the customization of the threshold angles for every finger and every user.
- Detecting an elementary tap: here, a basic tapping action is detected.
- Determining the tap position: when a tapping finger reaches its deepest point, various features are extracted such as its position.

- Isolating the tapping finger: since the human fingers move together, the tapping finger had to be correctly separated from the other moving fingers.

The remaining step is detecting the tap and generating an event for it.

A midair tapping gesture occurs when a finger goes in a Down-Up motion, separated into Click Down and Click Up steps. Since the human fingers move together, there is a possibility of detecting clicks that are unintentional; this is why we have isolated the intended finger. A result of the simultaneous motion of multiple fingers is that there are going to be a Click Down and a Click Up events for every finger that has crossed its threshold. Considering that a tap is a Click Down followed by a Click Up, once the instructed finger has been isolated, the other fingers' events should be rejected. To realize this, whenever a finger that has previously entered a Click Down mode goes back behind its threshold (i.e. becomes in Click Up mode), we check whether it is the previously isolated tapping finger. If it is, we suppose that this finger has satisfied both the Click Down and Click Up steps, and as a result, we generate its corresponding Click event. If it is not, no action is taken.

Since multiple fingers can move together, after the isolated finger has gone back behind its threshold and its relevant click event has been generated, there is a chance that the other fingers (the uninstructed fingers that moved simultaneously with the intended finger) are still crossing their thresholds. As a result, a Click Down event will be triggered for the deepest finger, and then a Click event will be generated for it once it goes back behind its respective threshold. This scenario is illustrated in Figure 3.3 - c; here, the tapping finger was the little finger, and even though it has backed behind its threshold, the ring finger is still crossing it, and thus is still in a tapping position. To prevent those unintentional taps from being detected, we have added the following rule: once a tap has been generated, a new tap cannot be detected unless *all the fingers* have backed up behind their thresholds. This is used to further prevent new taps from being detected during the Click Up phase. Figure 3.3 illustrates the complete cycle for a tap detection. The complete algorithm is summarized in Algorithm 5.

3.5 Midair Keyboard

Our Xpli Tap algorithm not only has the potential of detecting midair taps for every finger, but it can also extract the precise position of the tapping gesture. To show the potential of our system, we have implemented a midair keyboard that uses our tap detection

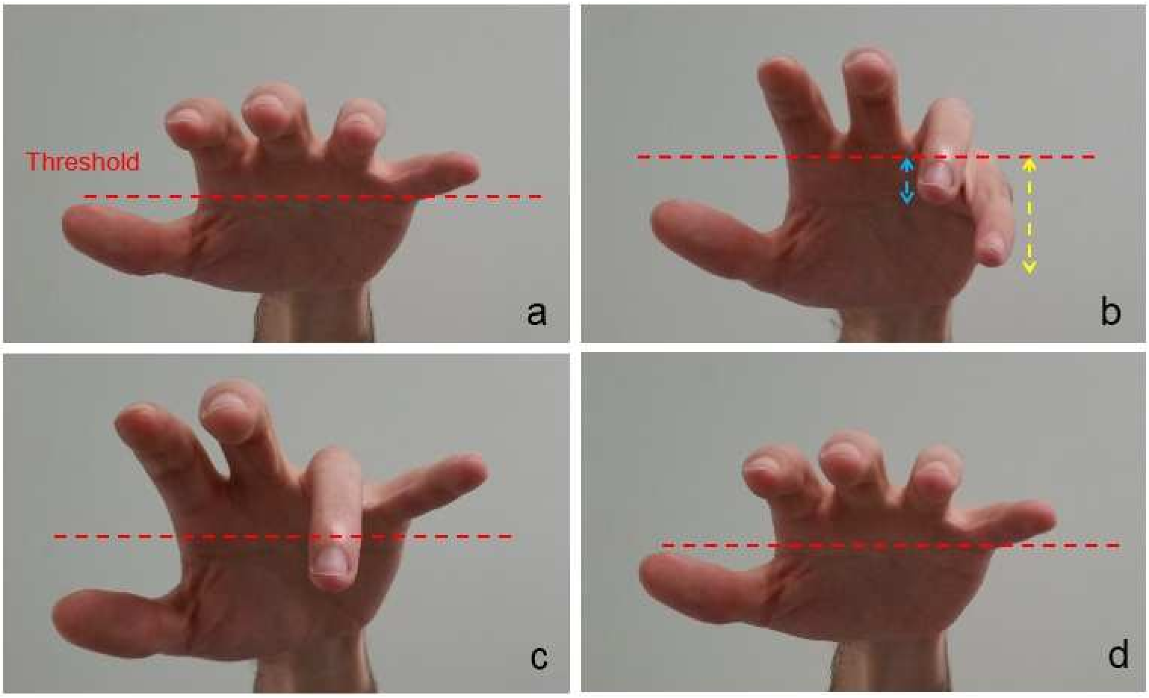


Figure 3.3: This figure depicts a little finger tap. In a, all the fingers are behind their thresholds. For the sake of simplicity, we have represented the threshold as a single red dashed line, whereas in reality, each finger has its own defined threshold. In b, both the ring and little fingers have crossed their thresholds, however, the Y-Travel of the little finger (in yellow) is bigger than that of the ring finger (in blue), so the little finger is detected as the tapping finger. The little finger has gone back behind its threshold in c, so a little-finger-click is generated, but since the ring finger is still crossing its threshold, no new tap can be detected (canClick == false). In d, all the fingers went back behind their thresholds, so a new tap can be detected hereafter.

mechanism. To evaluate it, we have defined the following three levels for the recognition rate:

- Level 0: 90% accuracy.
- Level 1: 99% accuracy.
- Level 2: 99.9% accuracy.

We have also defined the following 3 levels for input speed:

- Level 0: 8 strokes per minute - we can input in any way.
- Level 1: 40 strokes per minute - we can input with some stress.
- Level 2: 200 strokes per minute - we can input with no stress.

Our initial goals were Level 0 for accuracy and Level 1 for speed.

Algorithm 5 Tap Detection

```
procedure OnFrame
    for every Finger in hand do
        if (AngleOfFinger > $\theta$) then
            if canClick == false then
                return
            end if
            Find the tap position, then isolate the tapping finger
            CurrentFinger = tapping finger
            GenerateClickDown(CurrentFinger)
        else
            if (Finger has been in Click Down posture) && (Finger == CurrentFinger) then
                GenerateClickUp(CurrentFinger)
                canClick = false
            end if
        end if
        if All the fingers backed up behind their thresholds then
            canClick = true
        end if
    end for
end procedure
```

Previous work on midair keyboards has already been done. In [52], Yi et al. have created a midair keyboard. They have used a dictionary and a heuristic algorithm to estimate the words based on the sequence of fingers movements; that is, they analyze what fingers the users tapped, and generate a list of words depending on the possible combinations of keys that the fingers may have tapped. They also use one thumb to cycle through the possible results, and keep on using thumb clicks to go to the next word in the list of possible words that was generated from a dictionary. Once the desired word has been reached, the other thumb is then tapped to confirm it; a space is then appended automatically. This approach only uses combinations of keys and cannot be used to input individual keystrokes. Furthermore, it is impossible to input the space character by itself. Murase et al. [53] used a single RGB camera to detect key typing above a table. They estimate the row by detecting which row was selected. They accomplished this by calculating HOG features of the hands. While the required hardware is simple, the key press detection depends on two-dimensional values which prevents the users from freely twisting and turning their hands in 3D space. [54] used a projected keyboard on a surface, then estimated the pressed key by detecting the finger that touched the surface and mapped its coordinates with a calibrated projection matrix to find the required key. This solution requires a virtual keyboard to be projected on a surface which uses extra hardware and requires a physical space for the projected image.

Several attempts were made to create a virtual keyboard using Leap Motion. In [55], a QWERTY keyboard is projected on a virtual reality device, and users have to tap a key from that keyboard by having one finger intersect with an on-screen key. A similar approach is made in [56]. In both examples, users have to actually "hit" a key by hovering over it then tapping it, so their finger must be moved on top of the key before using it, therefore the hands must adapt to the keyboard's position. Moreover, it seems that only the index finger can be used to input keystrokes. A different method was used in [57]. Here, a QWERTY layout is not used, and the keys are arranged on a single line. From the above examples, we can say that no virtual keyboard has been realized yet.

By using Xpli Tap, we wanted to implement a midair keyboard that replicates as much as possible the operation of a physical QWERTY keyboard. For that purpose, we intended to implement a system that can explicitly detect each key by itself, without using any dictionary for finding the possible words that users may be trying to input. Our algorithm already detects taps from each finger. It also isolates the intended finger to make it more accurate and help preventing the input of the wrong key. However, additional work is required to create a midair keyboard: we must be able to distinguish the desired row. By detecting the clicking finger and the row, the key can then be determined. In the following paragraphs, we explain in details the assumptions behind the finger movements that occur when typing on a QWERTY keyboard, as well as the different methods we used to determine the row.

3.5.1 Fingers Movements Assumptions

We wish to implement a midair version of the QWERTY keyboard and keep it as close as possible, in terms of usage, to the physical one. In order to do that, we had to determine how the fingers moved when typists were utilizing the keyboard. We suppose that a keyboard is composed of three rows:

1. The upper row, containing the keys Q, W, E, R, T, Y, U, I, O, and P
2. The middle row, or home row, containing the keys A, S, D, F, G, H, J, K, L and Semicolon.

3. The lower row, containing the keys Z, X, C, V, B, N, M, Comma, Period and Forward Slash.

Overall, the keyboard is limited to the above mentioned keys. Furthermore, we consider that the space bar key is usable; however, we did not give it a “fixed” location below the lower row as is usually done in physical keyboards. The usage of the space bar is explained in 3.5.6.

In our study, we assume that when using a QWERTY keyboard, typists perform the following finger movements in order to type:

1. When they are not typing, the users’ fingers are over the home row. In this position, no tap is being performed and the fingers are in a relaxed position, ready to type.
2. To hit a key from the home row, the users perform a downwards tap from the relaxed position. We consider that no further movement of the hand is required to go from the relaxed position to tap a middle row key.
3. Regarding the index fingers, they are usually used to tap two different columns each. For example, the left hand finger is used to tap the keys “R, F, V” and “T, G, B”. Estimating the column is explained in details in 3.5.5.

As determined in the touch typing rules [58], we mapped the fingers, except the thumbs, to the keys as shown in Tables 3.1 and 3.2.

Table 3.1: Left hand fingers mapping.

	Little	Ring	Middle	Ring
Upper Row	Q	W	E	R or T
Middle Row	A	S	D	F or G
Lower Row	Z	X	C	V or B

Table 3.2: Right hand fingers mapping.

	Index	Middle	Ring	Little
Upper Row	Y or U	I	O	P
Middle Row	H or J	K	L	;
Lower Row	N or M	,	.	/

Successfully hitting a given key from the keyboard requires identifying the row as well. In order to accomplish that, we have defined three approaches:

- Tap Position Approach: the row will be determined based on the location of the tap.
- Finger Shape Approach: the shape of the tapping finger will be used to estimate the row.
- Palm Position Approach: the row will be approximated based on the palm's position.

The Tap Position Approach, the Finger Shape Approach and the Palm Position Approach are explained in details respectively in 3.5.2 , 3.5.3 and 3.5.4.

3.5.2 Tap Position Approach

We have previously discussed how our system is able to detect the exact position of a tap by determining the finger's deepest point during its motion (3.4.4). By using this feature, we can measure the distance from the palm center to the fingertip when a tap takes place; from that distance, we can approximate the row of the keyboard when a tap occurs. To realize this, we used the data collected in 3.4.2 and 3.4.4. To detect the row of a given tap, we have first defined $Distance_{MiddleRow}$ as 60% of the neutral palm distances of the index, middle and ring fingers, and 40% in the case of the little finger. We have then defined a margin of 15% for all the fingers. Once a tap is detected, we measure the new palm-to-fingertip distance, and use it in the following way, along with $Distance_{MiddleRow}$ and margin, to approximate the row:

- If the distance is smaller than $Distance_{MiddleRow}$ + *margin*, or greater than $Distance_{MiddleRow}$ - *margin*, then we assume that the finger hit the middle row of the keyboard.
- If the distance is greater than $Distance_{MiddleRow}$ + *margin*, we determine that the finger tapped the upper row.
- On the other hand, if the distance is smaller than $Distance_{MiddleRow}$ - *margin*, then we suppose that the finger tapped the lower row.

By using this approach, each finger tap has now three functionalities, rather than just one. We have thus been able to augment the tap's interaction vocabulary. Figure 3.4 illustrates how a row is determined in the case of the left hand's index finger.

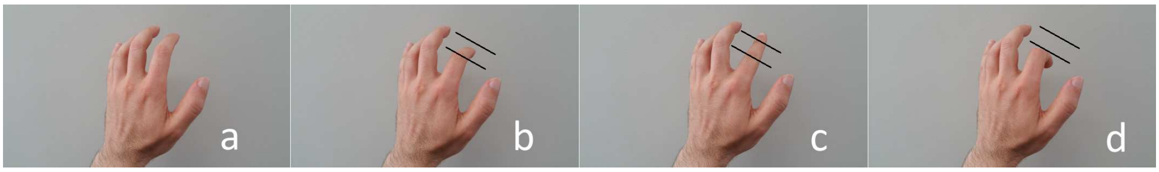


Figure 3.4: Illustration of how a row is estimated, in the case of an index click. The two parallel black lines represent the margin area for the finger. In a, the hand is in the neutral position. b shows the click occurring inside the margin, and thus the middle row is detected. In c, the click takes place beyond the margin, which means the upper row was clicked. The bottom row was detected in d, since the click occurred below the margin.

3.5.3 Finger Shape Approach

Just like Tap Position Approach (3.5.2), this method is used to increase the possible vocabulary of a single tap: instead of just assigning one given action to a tap, we detect the finger's shape then assign an action to *each* shape upon a successful tap detection. Therefore, instead of using the tap's positions, we will use the finger's shape during the tapping action to detect the row. Since our keyboard has three rows, we need to detect three different finger shapes in order to accomplish this task. We have defined shapes that occur while performing a tap: "Extended", "Bent" and "Relaxed". Figure 3.5 shows the different shapes of a tapping index finger.

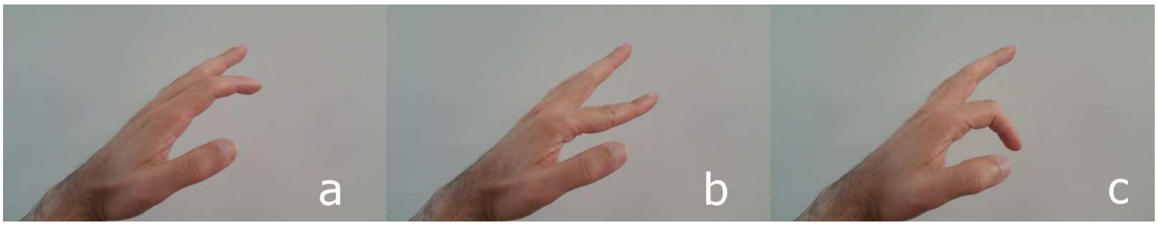


Figure 3.5: Illustration of the different finger shapes that we detect during a tap - here we show a tapping index finger. In a, the finger is relaxed. In b it is extended, whereas in c it is bent.

An extended finger is almost straight; we assume that this shape will occur when users are trying to tap the upper row of the keyboard. A bent finger will have its bones at an angle of approximately 90°; this is the shape that is supposed to be used when hitting the lower row. A relaxed finger is neither straight nor bent. This is the shape of a finger when in neutral position (3.4.1). If a finger has this shape when a tap is detected, then we assume that the intended row is the home row. To determine these shapes, we use the $angle_{bones}$

(Fig 3.6) extracted in 3.4.2. If, for a given finger, its $angle_{bones}$ is less than 30°, we suppose that it is extended. If it is greater than 90°, then it is bent. If a finger is neither extended nor bent, then it is relaxed. Those values were determined taking into consideration the gesture: the shape of the finger has to be varied without actually moving the hand. The value 90° was chosen as the shape in this case is easy to approximate by the user. An extended finger is straight, and will thus have its $angle_{bones}$ between 0° and 10°. To reduce errors, we chosen a value of 30°. Upon detecting a tap, we then perform the above check to determine the shape of the finger; this helps assigning extra functionalities to a tapping action.

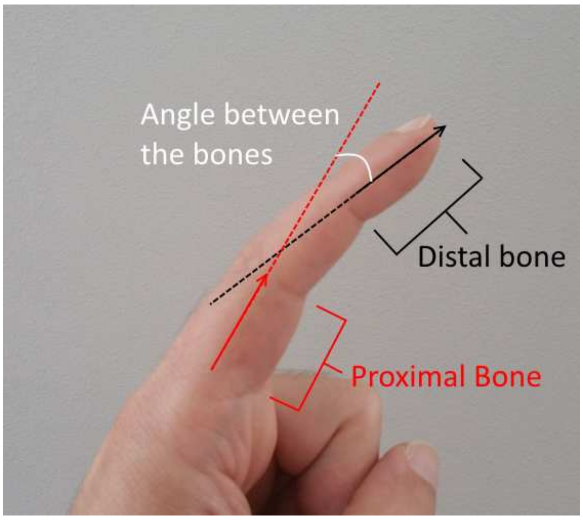


Figure 3.6: The angle between the distal (black) and proximal (red) bones of the index finger.

3.5.4 Palm Position Approach

In the previous methods (3.5.2, 3.5.3), the interaction vocabulary of a single tap has been augmented by relying on the tap position or the finger shape, that is, even if the hand did not move, the different augmented actions could be detected. In this technique, increasing the vocabulary relies on actually moving the hand before executing a tap. The position of the hand (that is, the position of the palm center) will be used to differentiate different actions for a single tap. To map this technique to our keyboard, we assume that typists move their hands to hit different rows. We therefore consider the following motions:

1. For an upper row key, they slightly move their hands away from their body before performing a tap.
2. In the case of a lower row key, they slightly move their hands closer to their body, then tap.

We have already retrieved $Palm_{neutral}$, the palm center in the neutral position (3.4.2). When a tapping action occurs, we determine the position of the palm center $Palm_{tap}$. To estimate the row that the users want to input, we compare $Palm_{tap}$ against $Palm_{neutral}$. To accomplish this, we compute the distance value of $[Palm_{tap}, Palm_{neutral}]$ according to the Z coordinates only, that is, if the hand moved on the X axis (left-to-right or right-to-left motion) or relatively to the Y axis (vertical motion), the results are not affected. We then compare that distance against a predefined threshold (we used 60 mm in our study). If it is less than the threshold, we consider that the tap occurred in the middle row. If not, the tap then occurred in the upper or lower row. In this case, we check the Z coordinates of $Palm_{tap}$ and $Palm_{neutral}$: if $Palm_{tap}$ is beyond $Palm_{neutral}$, then we suppose that the tap occurred in the upper row region. Conversely, if $Palm_{tap}$ is below $Palm_{neutral}$, then we suppose that the tap occurred in the lower row region. The Leap Motion sensor has a 150° left/right field of view, a 120° front/back field of view, and a range of 600 mm. Furthermore, it has an "interaction box", a rectilinear shape in this field of view in which the hand (or finger) is guaranteed to be detected. The Leap Motion controller software allows the modification of the size of this interaction box. By setting the box's height higher, its size becomes larger and the usable depth became 200 mm (100 mm on each side of the sensor according to the Z coordinate). By taking those values into account, as well as the to-and-fro movement of the palm, we have empirically determined the above mentioned 60 mm threshold to minimize the generation of errors. Figure. 3.7 illustrates the different zones that we defined in our row detection approach.

3.5.5 Column Estimation

As defined in Tables 3.1 and 3.2, each index finger can be used to input keys from two different columns: "R, F, V" and "T, G, B" for the left hand, "U, J, M" and "Y, H, N" for the right. We call "T, G, B" and "Y, H, N" the outermost columns. On the other hand, "R, F, V" and "U, J, M" are called the innermost columns. When typing a key from the outermost columns, the users extend their index fingers horizontally (away from the middle

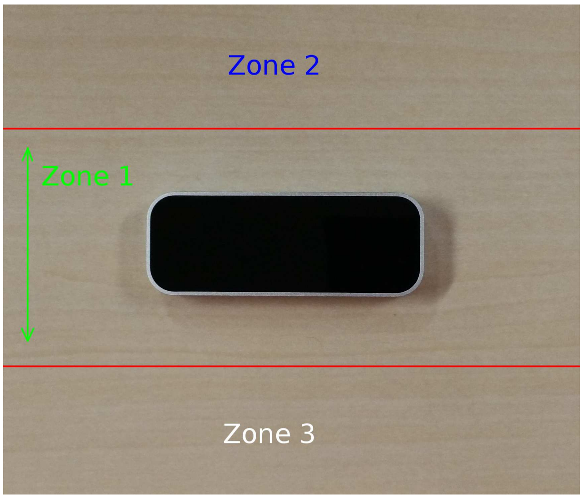


Figure 3.7: The different zones that are detected to determine the row. Zones 1, 2 and 3 are respectively mapped to the middle, upper and lower rows.

finger) in regards to their palm before executing a tap. To estimate the column tapped by an index finger, we used the angle $\angle\ MPI_{neutral}$ computed in 3.4.2. To detect the column, we suppose the users will move their index away from the middle finger and closer to the thumb, and thus, the new $\angle$ MPI angle will be greater than $MPI_{neutral}$, so we defined the threshold $MPI_{threshold} = MPI_{neutral} + 6°$. During a tap, we measure the new value of $\angle$ MPI; if it is greater than $MPI_{threshold}$, we suppose that the outer column was hit ("T, G, B" in the case of the left hand, "Y, H, N" in the case of the right). Conversely, if it is smaller than $MPI_{threshold}$, we consider that the innermost columns are tapped.

3.5.6 Midair Keyboard Implementation

In our research, we suppose that the users will utilize the midair keyboard in the same way they do a physical QWERTY keyboard. The keys are assigned to fingers in the same way too. For example, the little finger of the left hand can tap the "Q, A or Z" keys. The possible keys on our keyboard are the twenty-six alphabet keys, in addition to the following four keys: semicolon (;), comma (,), period (.) and forward slash (/). The space key is also usable. We consider the following steps in detecting a tap:

- If a thumb tap was detected from either hand, a space key tap is generated, regardless of the position of the tap.

- If a middle finger, ring finger, or little finger tap was detected, we deduce the row of the tap, then generate a tap of the corresponding key.
- In the case of an index tap, we deduce both the row and the column (3.5.5). We then generate the appropriate key tap.

In order to estimate the keys using the above steps, we used the data collected in 3.4.6. Knowing the tapping finger, the hand, the row (and column, in the case of an index finger), it is easy to determine the key. For example, if the right hand's index tapped the upper row and outer column, then the key is Y. Figure 3.8 shows the onscreen rendering of the midair keyboard. As explained in 3.6.2, the onscreen keyboard is used for feedback purposes only, that is, its shape and size do not affect the detection. And even though the onscreen keyboard looks "rectangular", during detection the keys are actually distributed along the fingers themselves, creating a more ergonomic use. And since the detection is performed relatively to the palm of each hand, the hands become fully independent of each other in terms of position and rotation: they do not need to be on the same level, and can be rotated in any position that is comfortable to the user.

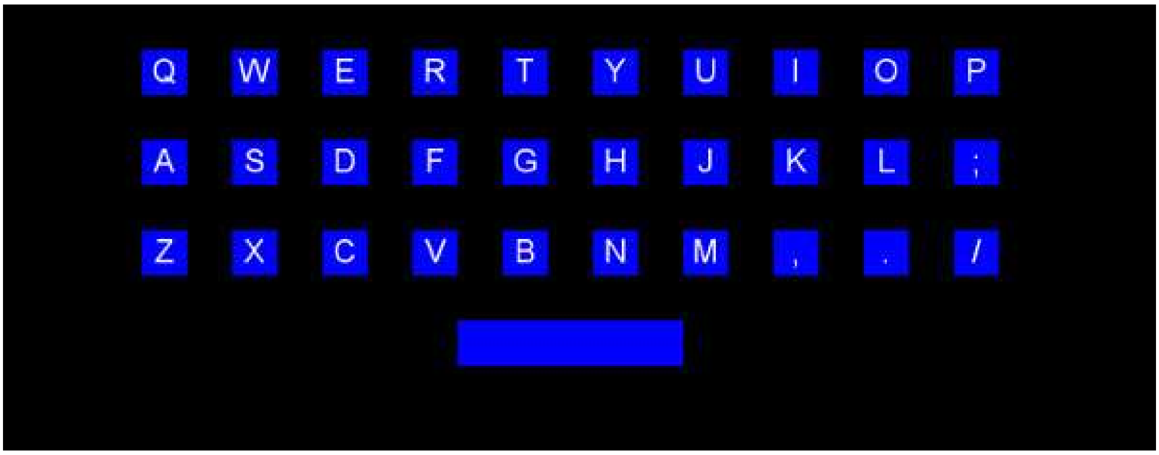


Figure 3.8: Onscreen rendering of the midair keyboard.

3.6 System Overview

In this section we describe the various hardware that we used for our system, as well as the prototype implementation.

3.6.1 Hardware

To implement our prototype, we used a Leap Motion sensor, which runs at approximately 100 fps (the frame rate varies depending on the available resources). A 23-inch monitor with Full HD resolution was used as the output device. The prototype has been implemented on a computer equipped with an Intel Core i5 3.2 GHz CPU and 4GB of RAM.

3.6.2 Prototype

We used the Leap Motion SDK [59] to detect the hands, as well as to retrieve the 3D positions of the fingers, palm center, and the normal vector perpendicular to the palm. We used those points to construct the various vectors, angles, and distances described in 3.4.1, 3.4.3, 3.4.4, 3.4.5 and 3.4.6.

Leap Motion allows the use of absolute and relative coordinates systems (Figure 3.2). We have chosen the relative coordinate system for the greater precision and flexibility that it allows, since the hands can become totally independent of each other in this case. For example, the hands can be on different height levels, or turned inward in different angles. Onscreen rendering was implemented in SFML [36], and the entire prototype was written in C++. The prototype's interface consisted of an onscreen keyboard containing thirty keys of a QWERTY keyboard: the twenty-six letters of the alphabet, plus the extra four keys as described in 3.5.6. The space bar was also included. The keys' background was rendered in blue, and the text in white. The tap detection algorithm would recognize taps and send the information to the onscreen keyboard. Whenever a key was tapped, its background color changed briefly to green and then back to blue; this provided the users with visual feedback.

3.7 Evaluation

Here we evaluate the Xpli Tap algorithm by using our midair keyboard. We have developed three different approaches for detecting the rows; we therefore evaluate them separately. For each approach, we explain in details the experiments and we discuss the results.

3.7.1 Tap Position Approach Experiments

This evaluation part is related to Tap Position Approach (3.5.2). To evaluate our system, 14 participants (13 males, 1 female) aged between 21 and 28 were recruited; 13 among them were computer scientists/engineers. All of them had to be able to touch type. We carried out three experiments to analyze the accuracy of Xpli Tap. The experiments consisted of using the midair keyboard for either inputting all keys, each key five times, or inputting several words.

A brief introduction was given to the participants on how to carry out the experiments. They were then given ten minutes to familiarize themselves with the tap detection algorithm. One experiment (three tasks) took approximately 30 minutes to complete. Participants 7, 9 and 13 were allowed to practice for twenty minutes instead of ten. The experiment apparatus consisted of a Leap Motion sensor placed on a desk, and a computer screen on which the participants could see the results of what they were typing. To perform the experiments, the participants placed their elbows on the desk, and their hands about 20 cm above the Leap Motion sensor, as can be seen in Figure 3.9.

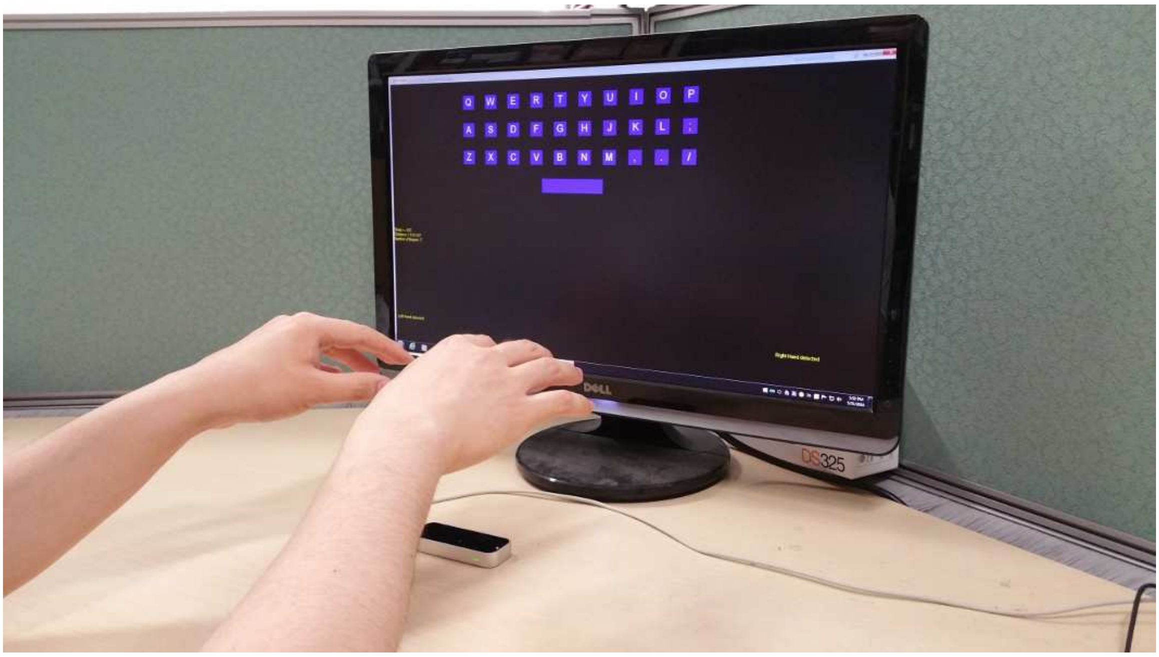


Figure 3.9: Midair keyboard prototype.

The first experiment consisted of entering every key (the twenty-six letters of the alphabet, plus the extra four keys) five times. The second experiment consisted of inputting nine words consisting of keys from all three rows of the keyboard. The third one is similar to the second, where the participants had to input eight words; this time, however, the words consisted only of keys from the upper and middle rows of the keyboard. We explain the experiments in details in the following subsections.

3.7.1.1 Experiment 1

In the first experiment, the participants were asked to input all thirty keys, five times each (FFFFF, RRRRR, VVVVV...). They started with the index finger of their left hand (F, R, V, T, G, B) then moved to the middle (D, E, C), ring (S, W, X) and little (A, Q, Z) fingers. They then repeated the same pattern with their right hand's index (J, U, M, H, Y, N), middle (K, I, comma), ring (L, O, period) and little (semicolon, P, forward slash). The purpose of this experiment was not only to determine the detection and precision of Xpli Tap, but whether repeated input of the same key led to a better detection (that is, whether training can improve the participants' accuracy). Since in this experiment, the participants had to input the same key five times, if the input was incorrect, they had the chance to readjust the position of their fingers depending on the output, and re-attempt to input the key during the remaining number of times. In this experiment, the participants made 150 input.

3.7.1.2 Experiment 2

In the second experiment, the participants had to input the following words: "unix cuba; cavity/ proxy voice maze aztec bump. cozy". They were also required to input spaces between every two words. The above word list covers all rows from the entire keyboard; all fingers had to be used too. While performing this experiment, the participants had only one chance per key; if the input was incorrect, they did not have the possibility to readjust their fingers and re-input the key. In this experiment, the participants made 52 inputs.

3.7.1.3 Experiment 3

This experiment is similar to Experiment 2; however in this case, the word list consisted of words made only from all the keys of the middle and upper row of the keyboard; the words still consisted of letters that covered all the fingers. The following words were used: "fish ugly dial dusty hook; powerful walk qed". As in Experiment 2, the users had only a single attempt to hit the correct key. Here, they made 44 inputs.

3.7.1.4 Questionnaire

After completing the experiments, the participants were asked to fill out the following questionnaire, to which they could respond on a five point Likert scale (-2 = Strongly negative answer, 2 = Strongly positive answer):

1. In what way does the keyboard made with our tap detection replicate a keyboard's usual operation?
2. To what extent does our tap detection's usage feel natural?

The participants were also offered the opportunity to enter comments, if any, freely.

3.7.1.5 Results

Each participant made 150, 52 and 44 inputs in Experiment 1, Experiment 2 and Experiment 3 respectively, for a total of 246 inputs. All in all, 3444 inputs were made by 14 participants. We have only counted the results that were the same as the intended keys. Simultaneous key detection was rejected: if more than one key was outputted for a single tapping gesture, it was counted as a false detection.

Figure 3.10 shows the results of the exact hits detection average. The results of all the keys, except the space bar, are deduced from Experiment 1. The results of the space bar accuracy are deduced from Experiments 2 and 3. Overall, this approach had a detection rate of 68.9%.

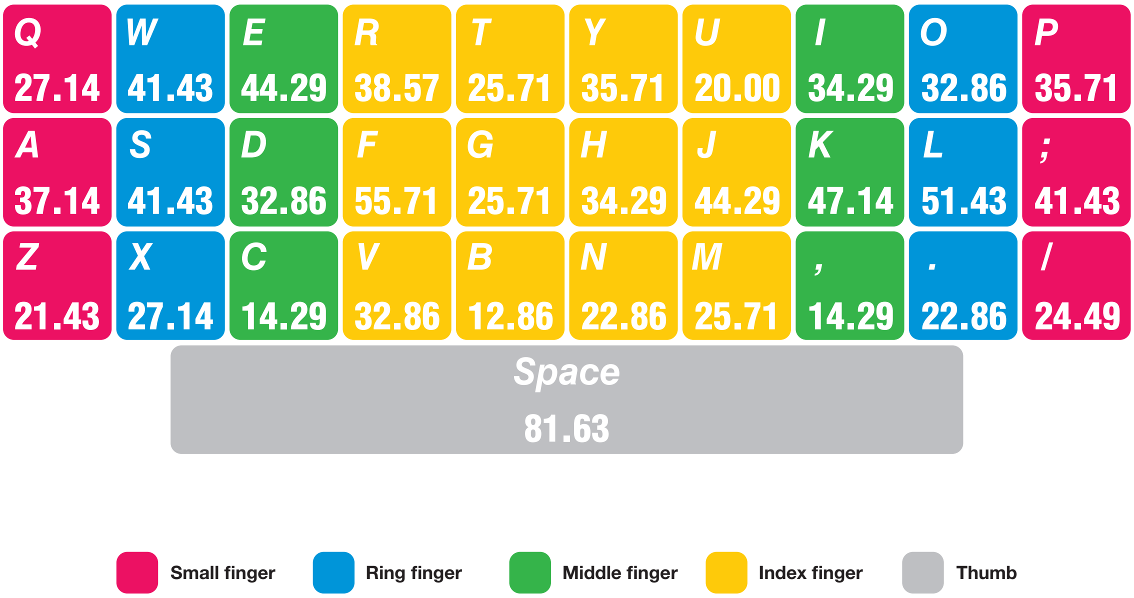
The results of the questionnaire are shown in Table 3.3

Table 3.3: Questionnaire results.

Question	Mean	SD
Question 1	-0.29	0.73
Question 2	0.43	0.85

3.7.1.6 Discussion

We will start the discussion by evaluating the tap detection. We will then elaborate on each experiment individually. While the experiments where being performed, many taps



Q 27.14	W 41.43	E 44.29	R 38.57	T 25.71	Y 35.71	U 20.00	I 34.29	O 32.86	P 35.71
A 37.14	S 41.43	D 32.86	F 55.71	G 25.71	H 34.29	J 44.29	K 47.14	L 51.43	; 41.43
Z 21.43	X 27.14	C 14.29	V 32.86	B 12.86	N 22.86	M 25.71	, 14.29	. 22.86	/ 24.49

Space
81.63

Figure 3.10: Results of the Tap Approach Detection Average.

went undetected. We attribute this to two reasons. The first reason is that the tap detection depends on the calibration step introduced in 3.4.2. The participants had to calibrate before each training session, and each experiment step. Therefore, there is a possibility that the participants could not replicate the same calibration throughout the entire experiment. In some cases, for the same participant, the training phase had a high detection rate which could not be reproduced in the real experiments. In other cases, the calibration was effective for just one hand, and thus the detection rate was high for said hand, and low for the other one. The second reason is the tap detection condition proposed in 3.4.6. In this condition, after a tap has been performed with a finger, all the fingers are required to go back behind their thresholds, otherwise no tap is detected. As the humans fingers move together [48], [49] any finger could have moved unintentionally during the tap; if it crossed its given threshold, it will prevent the intended tap from being detected.

Moreover, we discuss some factors that have contributed to lowering the accuracy results. We divide these factors into two categories:

1. The wrong finger was detected.
2. The right finger was distinguished, but a wrong key was detected.

For category 1, there are times where the Y-travel for the intended finger was less than the Y-travel of other finger(s), thus detecting the tap of a different finger. Participant 3 had his ring finger always engaged in tapping position, therefore generating unintended ring finger taps, whereas Participant 8 was unconsciously tapping with the thumb of the opposite hand while performing the taps. For category 2, most of the participants complained that it was very difficult to type in midair without physical feedback, and could not precisely hit the intended key. This is also due to how the algorithm was constructed: the difference in positions between the rows was subtle and difficult to be precisely hit. In some cases, the participants' finger motion also affected the results; for example, when Participant 5 was performing a little finger tap, his little finger was always fully extended, and as a result, only the upper row was being detected.

In Experiment 1, a single key was inputted five times. From the results, we have observed that many of the correct hits were the fourth and fifth hits. We assume that the participants were able to adjust their fingers positions based on previous inputs of the same key, and thus generating an exact hit. Participants 7, 9 and 13, who were given an extra time to practice, scored respectively 42.66%, 45.33% and 52%, higher than the mean of 32.19%. This suggests that with practice, the users can ameliorate the exact hits rate. This experiment's detection rate is also higher than that of Experiment 2, further corroborating this fact. While the detection rate of Experiment 3 was greater, we duly note that in Experiment 3, only the middle and upper row of the keyboard were used.

In the case of Experiment 2, the exact hits rate was lower than in Experiment 1. One of the reasons is that the users had only one chance to hit the exact key. Another reason was the fact that the keys belonging to the lower row of the keyboard were very challenging to hit. Many participants commented that the lower row was difficult to hit. This can be due to the motions used by the participants while performing a lower row tap. While the difference between the middle and lower rows is subtle and thus difficult to estimate, most of participants bent their fingers towards their palms in an exaggerated rate, which prevented a tap from being detected at all, thus contributing to the reduction of the detection rate.

Since the exact hits rate of this Experiment 3 is higher than that of Experiment 2, and considering that no keys from the lower row of the keyboard were used in this experiment, this suggests that in our algorithm, a tap of a key belonging to the middle or upper row was easiest to perform. A possible reason for this is the fact that when the participants tried to input a key from the lower row, their finger became very close to the palm and the sensor could not detect it anymore. This experiment had also a low no-detection rate, further validating the point raised in the analysis of Experiment 2.

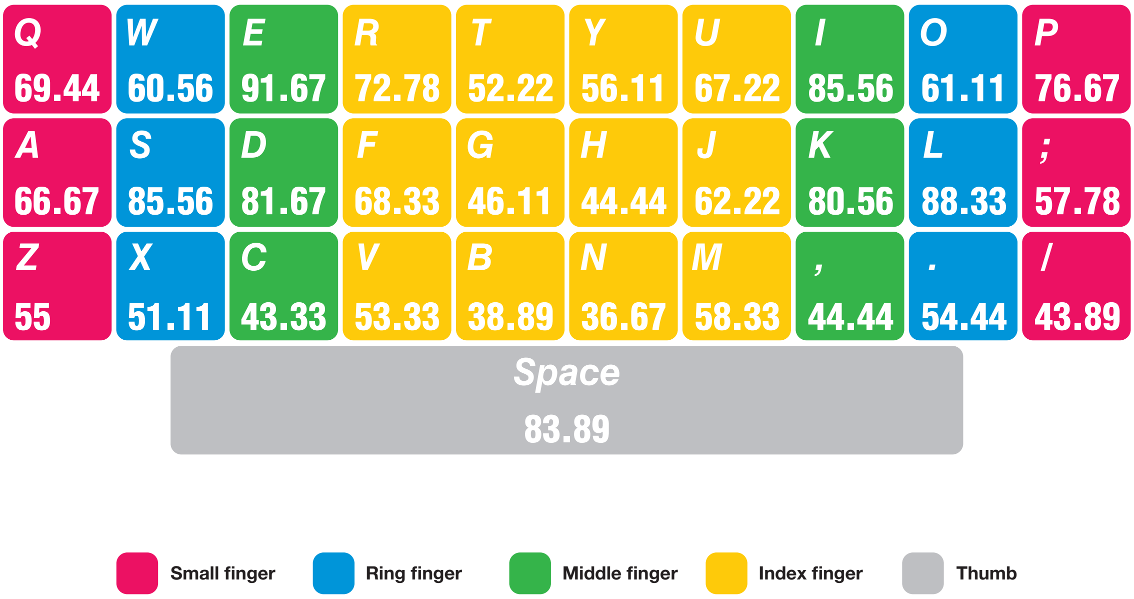
We now discuss the results of the questionnaire. The result of Question 1 shows that the participants did not agree that our algorithm could replicate a physical keyboard's operation. As a matter of fact, most of them did comment that typing in midair was too difficult. The result of Question 2 shows that the participants marginally agree that our tap detection's usage felt normal.

3.7.2 Experiment - Finger Shape Approach

In this section, we evaluate the Finger Shape Approach (3.5.3). 9 participants (all males) aged between 21 and 27 were recruited; 6 amongst them were computer science students. A brief introduction was given to the participants on how to carry out the experiment. They were given one hour to practice using the system. To perform the experiment, the participants placed their elbows on the desk, and their hands about 20 cm above the Leap Motion sensor, as can be seen in Figure 3.9. Our system consists of detecting midair taps, and identifying the indented tapping finger from uninstructed ones. Then a midair keyboard was implemented as an application for the tap detection algorithm. As a preliminary evaluation to our system, we have focused on evaluating the tap detection more than evaluating the keyboard. In this task, the participants were asked to input all thirty-one keys, twenty times each (QQQ..., WWW..., EEE, ...). They started with their left hand, and inputted the keys corresponding to the upper row of the keyboard (Q, W, E, R, T), then moved on to the middle row (A, S, D, F, G). Finally, they continued with the bottom row (Z, X, C, V, B). Next, they repeated the same task using their right hand, where they started with the upper row (Y, U, I, O, P), middle row (H, J, K, L, ;) and bottom row (N, M, comma, period, forward slash). Finally, they entered the "space" key twenty times. Overall, the participants made 620 inputs per trial. In the future, an evaluation focusing on the keyboard can be made in which random keys appear which participants would then attempt to hit, rather than hitting the same key repeatedly.

3.7.2.1 Results

Each participant made 620 inputs in the experiment. All in all, 5580 inputs were made by 9 participants. We have analyzed the success rate of the taps: we have only counted the results that were the same as the intended keys. Any simultaneous key detection was rejected: if more than one key was outputted for a single tapping gesture, it was counted as a false detection. Figure 3.11 illustrates the success rate by key.



Q 69.44	W 60.56	E 91.67	R 72.78	T 52.22	Y 56.11	U 67.22	I 85.56	O 61.11	P 76.67
A 66.67	S 85.56	D 81.67	F 68.33	G 46.11	H 44.44	J 62.22	K 80.56	L 88.33	; 57.78
Z 55	X 51.11	C 43.33	V 53.33	B 38.89	N 36.67	M 58.33	, 44.44	. 54.44	/ 43.89

Space
83.89

🟥 Small finger 🟦 Ring finger 🟩 Middle finger 🟨 Index finger ⬜ Thumb

Figure 3.11: Success rate (%) by key of Xpli Tap's Finger Shape Approach.

3.7.2.2 Discussion

The Finger Shape Approach and the Tap Position Approach (as well as the Palm Position Approach) share the same basic tap detection algorithm, which has already been discussed in 3.7.1.6. Therefore, we only discuss here the different aspect of the algorithm, which is using the finger shape to approximate the keyboard's row.

We first start by discussing the results of the lower row of the keyboard. As can be seen in Figure 3.11, compared to the middle and upper rows, the algorithm had the lowest accuracy in the lower row. This is mainly due to the inability of the participants to correctly tap keys from the lower rows. To accurately detect a key tap, our algorithm must detect both the tapping gesture, as well as the finger shape. If any was missed, the result will be invalid. To hit a key belonging to the lower row, the participants had to bend their fingers to about 90° (Figure 3.5 - c). This gesture was particularly challenging for the participants, as they had trouble performing a tap while bending their fingers in this shape. This has reduced the accuracy of the detection. Furthermore, we noticed that the angle of the finger bones was defectively reported at times. To detect this angle, we rely on the Leap Motion

SDK's ability to detect the vectors of the different bones that constitute the finger. Even though the fingers angles were clearly greater than 90°, the SDK reported an angle of about 20°-30°. This has caused erroneous key detections to occur and as a result decreased the accuracy of the algorithm.

This kind of angle detection error rarely occurred for the upper and home rows. Here, the fingers had to be either extended or relaxed to respectively tap the upper and middle rows. The likelihood of unsuccessfully detecting the fingers angles was low in this case. However, the participants reported that they found difficulties in differentiating the rows. The differentiation between the upper and home rows was subtle, which caused erratic results. This was mainly noticeable with the Little finger. The Ring and Middle fingers had overall higher accuracy in this case.

We discuss the Index fingers results independently as the unlike the other fingers, each Index finger can be used to tap keys from two different columns. When compared with the other digits, the Index fingers had lower results, regardless of the row. This is due to the column differentiation algorithm, where we measured the angle between the Index and Middle fingers. To implement the algorithm, we yet again relied on Leap Motion's SDK to retrieve the vectors representing those fingers. We have noticed in many cases, that even though the Index and Middle fingers were widely open (with an angle clearly greater than the threshold), the returned angle was negative, as if the fingers were crossed. This was the main reason for the decrease of the Index finger' accuracy.

Furthermore, some wrong keys detections are due to the fingers movements as discussed in 3.7.1.6. The participants had to focus on keeping the tapping finger in a certain shape; many of them unconsciously kept other fingers in their tapping position, especially the thumb.

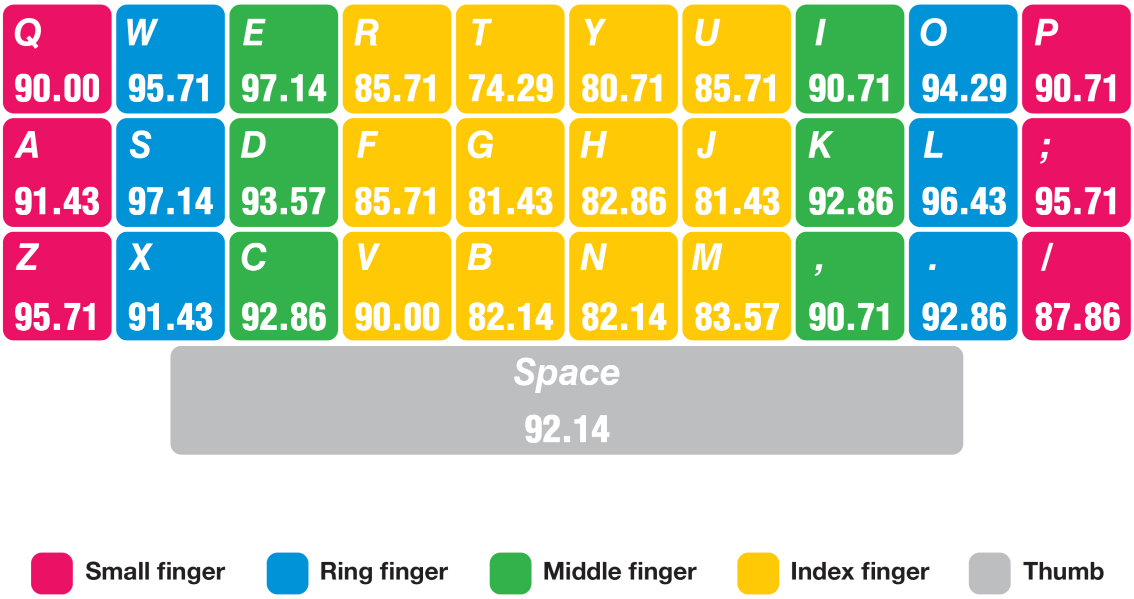
Moreover, we would also like to mention that the specifications of the sensor might influence the results. For example, a sensor using radar waves (impervious to ambient light) and with higher accuracy might generate better results.

3.7.3 Experiment - Palm Position Approach

Here we evaluate the Palm Position Approach. 14 participants (1 female) aged between 22 and 27 were recruited; 10 amongst them were computer science students. We carried out an experiment to analyze the accuracy of Xpli Tap's Position Approach. A brief introduction was given to the participants on how to carry out the experiment. They were given 2 hours to practice using the system. The participants were asked to input all thirty-one keys, ten times each (QQQ..., WWW..., EEE, ...). They started with their left hand, and inputted the keys corresponding to the upper row of the keyboard (Q, W, E, R, T), then moved on to the middle row (A, S, D, F, G). Finally, they continued with the bottom row (Z, X, C, V, B). Next, they repeated the same task using their right hand, where they started with the upper row (Y, U, I, O, P), middle row (H, J, K, L, ;) and bottom row (N, M, comma, period, forward slash). Finally, they entered the "space" key ten times. To perform this task, they placed their elbows on the desk, and their hands about 20 cm above the Leap Motion sensor, as can be seen in Figure 3.9, just as was done in the previous experiments.

3.7.3.1 Results

Each participant made 310 inputs. All in all, 4,340 inputs were made by 14 participants. We have performed the same analysis that we conducted in Experiment 1. We have analyzed the success rate of the taps: we have only counted the results that were the same as the intended keys. Any simultaneous key detection was rejected: if more than one key was outputted for a single tapping gesture, it was counted as a false detection. Figure 3.12 illustrates the success rate by key of for Xpli Tap.



Q 90.00	W 95.71	E 97.14	R 85.71	T 74.29	Y 80.71	U 85.71	I 90.71	O 94.29	P 90.71
A 91.43	S 97.14	D 93.57	F 85.71	G 81.43	H 82.86	J 81.43	K 92.86	L 96.43	; 95.71
Z 95.71	X 91.43	C 92.86	V 90.00	B 82.14	N 82.14	M 83.57	, 90.71	. 92.86	/ 87.86
		Space 92.14							

Figure 3.12: Success rate (%) by key of Xpli Tap's Palm Position Approach.

3.7.3.2 Discussion

In this approach, the position of the palm was used to discriminate between the rows. Moving the hand is easier than bending the fingers to a given shape or attempting to identify a location in midair. This has contributed in getting higher results as can be seen in Figure 3.12. In consequence, these results accurately reflect the performance of Xpli Tap, rather than that of the row detection. We therefore discuss the new results, without going back to the common points that have been discussed in 3.7.1.6 and 3.7.2.2.

As can be seen, Xpli Tap has an accuracy higher than 90% for most of the keys. The accuracy of the Ring and Middle fingers is also greater than that of the Little finger, just as reported in 3.7.2.2. Using this approach, the participants were satisfied compared to the previous two. The hindrance that they encountered was mainly the distinction of the Index finger columns; this too has been previously discussed in 3.7.2.2.

Xpli Tap has been shown to be a reliable midair tap detection algorithm, albeit having some decrease in accuracy when the keyboard's row and column detection constraints were introduced. In the future, we plan on using newer sensors which may detect the required positions more accurately and thus generate better results overall. Moreover, the column differentiation algorithm can be changed. As an example, a thumb finger tap may be used to detect the outermost Index columns, whereas an Index tap will be utilized to input keys from the innermost ones.

By checking the results of the Tap Position, Finger Shape and Palm Position approaches, we can learn that there is some kind of trade-off between precision and vocabulary: when using the tap position or finger shape to augment the vocabulary, the precision dropped, whereas it got higher when we used the palm position (which is simpler than the previous two in terms of adding functionalities). Future sensors with more advanced capabilities and higher sensitivity might help reduce that trade-off.

3.7.4 Input Speed Experiment

The results of the above experiments show that Palm Position Approach is more accurate than Finger Shape Approach and Tap Position Approach. We therefore have conducted the third experiment to determine the input speed of the Palm Position Approach. Here, the same participants from 3.7.3 were recruited. They were asked to input 5 sentences. Then, we measured the time required to input each of them. The following sentences were used:

1. when in rome, do as the romans
2. the pen is mightier than the sword
3. a penny saved is a penny earned
4. the quick brown fox jumps over the lazy dog
5. actions speak louder than words

In this experiment, we did not focus on the accuracy of the keyboard, but rather on the detection speed. Therefore, in case a mishit occurred, the participants would not correct it, but simply continue with the next character. The accuracy was also not used in determining the speed.

3.7.4.1 Results

Each participant made 168 inputs for a total of 2,352 for 14 participants. We measured the average speed in seconds it took the participants to complete the input of each sentence. We then deduced the input speed in characters per minute and we have obtained an average result of 57.43 characters per minute. Table 3.4 summarizes the results of the time (in seconds), speed (in characters per minute) and standard deviation for each sentence.

Table 3.4: Results of input speed.

Value	Sentence 1	Sentence 2	Sentence 3	Sentence 4	Sentence 5
Time [sec]	35.67	35.60	29.04	45.46	30.67
Speed [cpm]	50.46	57.31	61.98	56.75	60.65
S.D.	9.94	6.72	6.31	11.14	9.54

The expected speed depends on the application in which the system is being used. The results of the speed are a preliminary evaluation of a midair input mechanism. All the participants have spent years using a physical keyboard, and this was their first time using a midair keyboard, after only a limited amount of training time. Their transition from physical to midair is incomplete, and their usage of the keyboard is still influenced by the physical one. The obtained speed therefore falls within our expectation for the current situation.

3.8 Possible Applications

Our system allows the detection of taps from any finger of the hand, and uses also the finger shape of during the tapping operation to expand the interaction vocabulary. In this section, we describe some possible applications that take advantage of these features.

A possible application would be pairing our system with a dictionary and a heuristic algorithm. Depending on the used finger, its shape and position, the heuristic algorithm would then be able to analyze the sequence of taps to retrieve a list of most probable words from the dictionary, and suggest them to the users in order of decreasing probabilities. This would create a reliable midair keyboard than can be used as an effective input method for the above mentioned futuristic environments.

In the future, smartglasses and see-through head mounted displays such as the Atheer AiR Smart Glasses [60] or the Microsoft HoloLens [61] are expected to become increasingly common in our everyday lives. Google introduced "Soli", a gesture detection system that is small enough to fit in smartphones and smartwatches [62]. Such devices are some of the best examples of NUI, where users will be required to perform many gestures using their hands, and a tap detection mechanism will prove useful in many situations. A particular case where Xpli Tap might be advantageous is when it is used in an "eyes-free" environment. For example, a smartwatch has a very small screen which makes it difficult to display many clickable buttons in its interface. If it is equipped with a sensor like "Soli", it will be possible to assign functions not only to individual fingers, but to their shapes as well, thus increasing the interaction vocabulary of a single hand. And since the functions can be distinguished by fingers and their shapes, users can utilize them without the need to look at the screen.

DSLR cameras can also be a good candidate with which Xpli Tap can be useful. In some situations, a camera should not be operated with the hand, because touching it might shake it or change its direction (such as when using a long shutter speed while photographing fireworks). In this case, photographers use a cable release or remote control to trigger the camera's shutter. However, there is also a need to modify other functions of a camera, such as aperture, white balance or shutter speed. A camera equipped with a sensor can be operated without touching it; and by assigning functions to different fingers, it can also be operated in an eyes-free way. For example, this adds the possibility for a photographer to change some settings while taking a self-portrait without the need to look at the screen or visor of the camera.

3.9 Summary

In this chapter, we presented Xpli Tap, a novel approach to detecting midair finger taps. We used Depth Click from Chapter 2 as our starting point, and added multiple improvements to the algorithm. First, we added a new Relaxed Neutral Position during which the hands are relaxed and not performing any taps. Next we introduced a calibration step which allowed to detect multiple parameters related to the hand and fingers. Those values were used throughout the algorithm. We then introduced a routine in which the exact position of the tap was detected. Here, the motion of the tapping finger was analyzed; when the finger reached its deepest point, we considered that point as the tap position. Detecting the exact tap position allowed us to increase the interaction vocabulary of a single tap. We then combined the above mentioned steps to create a reliable midair tap detection mechanism. To increase its accuracy, we also included a new rule which prevents a tap from being detected unless all the fingers have backed up behind their thresholds. To show the advantages of Xpli Tap, we implemented a midair keyboard that relies on detecting the keys without using any dictionary. To detect the rows of the keyboard, we used three different approaches: the Tap Position Approach which uses the exact location of the tap to increase the gestures vocabulary, the Finger Shape Approach which relies on the finger shape to augment the interactions and the Palm Position Approach which in turn utilizes the palm's position during a tap to expand the interaction vocabulary. We have also conducted several experiments to evaluate the above mentioned approaches, and we deduced that the Palm Position Approach generates the most accurate results. From the results of these experiments, we can say that our initial goals, i.e., Level 0 for accuracy and Level 1 for speed, were satisfied in the Palm Position Approach. The results also show that when removing the row detection constraints which were added for the midair keyboard, Xpli Tap proved to be a reliable tap detection algorithm.

Chapter 4

A Method To Detect Midair Multi-clicks Gestures

4.1 Introduction

This chapter describes MultiX Click, a midair multi-clicks gestures detection method. First, we describe the benefits of multi-tap detection. Then we explain in details how we implemented a midair multi-click keyboard and the zones that we created to trigger multi-click detection. We later discuss another application of our method, a midair piano. We then conduct experiments to determine the feasibility of MultiX Click. Finally, we discuss the method and its limitations and we describe some other possible applications.

4.2 Research Goals

We have previously discussed the advantages of midair tap detection in Chapter 3. We have focused on isolating the tapping finger when multiple fingers move together to make the system more reliable. We have also detected the exact position of the tap and successfully used it to augment the gesture vocabulary whenever a tap is detected. Furthermore, we have detected the tapping finger's shape and effectively used it to further increase the vocabulary. We would like to enrich the gestures vocabulary even more. To accomplish this, we have implemented a multi-tap detection method that allows the simultaneous detection of multiple taps. Moreover, we created a midair keyboard with multi-click capabilities to detect a combination of modifier keys and regular keys. By definition, when using a keyboard, only one tap should be detected at a given time to prevent the multiple input of several keys. To overcome this, we create zones in which a multi-tap can be triggered. Additionally, we implemented an air piano in which users could simultaneously tap multiple fingers to play piano chords.

To assess our method, we have established some targets which should be reached. First, we have defined the following three levels for the recognition rate:

- Level 0: 90% accuracy.
- Level 1: 99% accuracy.
- Level 2: 99.9% accuracy.

We have also defined the following 3 levels for input speed:

- Level 0: 8 strokes per minute - we can input in any way.
- Level 1: 40 strokes per minute - we can input with some stress.
- Level 2: 200 strokes per minute - we can input with no stress.

The target recognition rate and the target speed will depend on applications. For the Midair Keyboard application, our initial goals were Level 1 for accuracy and Level 1 for speed. For the Air Piano application, our initial goals were Level 0 for accuracy and Level 1 for speed.

4.2.1 Related Work

Multi-touch interaction involves using more than one finger while operating a given interface. Multi-touch has been researched extensively. Colley et al. [63] report that users preferred the multi-finger technique to the traditional interface while utilizing a touchscreen.

In HandyScope [64], Kuribara et al. used two fingers to invoke a remote control mechanism for large tabletop setups, and a third finger to interact with the remote control.

Murugappan et al. [65] used a touchscreen to detect multi-touch and a depth camera to identify users, their fingers and handedness; mixing those allowed them to create rich interactions. A similar approach was used in [66] to differentiate the fingers that are being used to operate the touchscreen; they suggest that identifying the fingers can be helpful in some situations such as using shortcuts.

In SixthSense [67], Mistry et al. used markers on fingertips to detect the fingers and their gestures, including multi-touch gestures such as simultaneously detecting four fingers (two fingers from each hand) performing a “framing” gesture to trigger a camera.

Sridhar et al. [68] used midair multi-finger interactions for word input by differentiating fingers and assigning different characters depending on the simultaneously flexed fingers.

Multi-touch gestures were suggested to be a promising technique for biometric authentication in [69] where the authors detected twenty-two different multi-touch gestures.

4.3 MultiX Click

The importance of midair gestures has been discussed in Chapters 2 and 3. These gestures, however, have been executed using one finger at a time. The ability to simultaneously use more than a finger can notably augment the gestures' vocabulary. For example, different actions can be assigned to gestures that are using two fingers. Discriminating the fingers can further help increasing the vocabulary. Here, actions can be assigned to a number of fingers, as well as their identity.

In this section, we introduce MultiX Click, a method that we have developed to detect multi-clicks, that is, simultaneous detection of tapping gestures coming from different fingers. To implement MultiX Click, we used Xpli Tap (3.4) as our starting point. While Xpli Tap explicitly singles out the tapping finger when multiple ones move simultaneously (3.4.5), it does originally detect the taps from all the hand's fingers (3.4.3). This allowed us create different possibilities of interactions. We were able to mix finger isolation with multi-tapping. Furthermore, we were able to discard the finger singling part and created pure multi-tapping operations. Moreover, we have created two applications that use these new methods.

The basic tap detection algorithm is the same as the one used in Xpli Tap (3.4.6); for this reason, we do not have a section presenting the MultiX Click method. However, as we have created two applications for the method, and since the changes were introduced in the respective applications, the particulars of these implementations will be discussed in details in sections 4.5 and 4.6 to show the new functionalities that have been added to Xpli Tap to make it possible to detect multi-taps.

4.4 System Overview

In this section we describe the various hardware that we used for our system, as well as the prototype implementation.

4.4.1 Hardware

To implement our prototype, we used a Leap Motion sensor, which runs at approximately 100 fps (the frame rate varies depending on the available resources). A 23-inch monitor with Full HD resolution was used as the output device. The prototype has been implemented on a computer equipped with an Intel Core i5 3.2 GHz CPU and 4GB of RAM.

4.4.2 Prototype

To test our method, we have implemented a midair multi-click keyboard and a midair piano. We explain about them in details in 4.5 and 4.6 respectively. We used the Leap Motion SDK to detect the hands, as well as to retrieve the 3D positions of the fingers, palm center, and the normal vector perpendicular to the palm. We used those points to construct the various vectors, angles, and distances used throughout our method. Onscreen rendering was implemented in SFML, and the entire prototype was written in C++.

4.5 Midair Multi-click Keyboard

To test MultiX Click, we have implemented a "midair multi-click keyboard", which can be used to input the alphabet characters, some special keys, as well as the "Shift", "Control" and "Alt" modifier keys. In a physical QWERTY keyboard, the modifier keys are used to modify the input that is usually generated when using a normal key. For example, the Shift key is usually used to output a capital letter when it is used in tandem with an alphabet character. For example, to input the character "A" (capital A), the combination "Shift + A" is used, that is, the Shift key is tapped and held down, then the A key is tapped, and finally, the Shift key is released. This operation can be generalized in the following manner:

1. Tap and hold down a modifier key.
2. Tap a character.
3. Release the modifier key.

As a general rule, the hand used to input the modifier key is usually the opposite of the hand that is being used to input the regular character. In the case of "Shift + A", since A is inputted with the left hand, the Shift key has to be held down then with the right hand. In our midair multi-click keyboard, we wish to replicate this operation.

In a physical keyboard, more operations are possible. For example, the Shift key can be used with the numeric key “1” to output an exclamation mark (“!”). Moreover, multiple modifier keys can be combined with a regular character (“Ctrl + Shift + P”). In our study, we will only limit the simultaneous detections to one modifier key and one character belonging to any of the basic three keyboard rows.

4.5.1 Triggering the modifier keys

A problem arises when we are mixing both multi-clicks interactions with single-finger taps interactions; in some situations, multiple functionalities might overlap if the same finger is used to detect a Click Down, and later is detected as a regular tap once it goes back behind its threshold. While our system supports this kind of gestures, we have designed the single-finger clicks and multi-clicks to be mutually exclusive, and only one of them can be used at a given time. To realize this kind of multi-click interactions that can be mixed with single-tap ones, we have defined two zones: the single-click zone, and the multi-click zone. When the hand is inside the single-click area, any tap will be considered a single-tap, that is, the finger isolation will be activated and even if multiple fingers move simultaneously, only one tap will be detected. Whenever the hand moves to the multi-click area, then multiple taps can be used concurrently, thus making it possible to input combinations of a modifier key and alphabetical key. To accomplish this, we have defined a circle-shaped threshold with the sensor as its center (single-tap zone), and the multi-clicks detection will only be activated if the users’ hands are outside the circle (that is, if the distance from the palm center to the sensor is greater than the circle’s radius). In our system, we used a radius of 80 mm; using a bigger value will increase the chance of having the hand outside the detection range of the sensor, and will hence decrease its accuracy. To further increase the vocabulary of the interactions, we used the location of the hand’s palm center to determine its relative position to the sensor, in the horizontal plane. To accomplish this, we measure the angle ∠ XOP between vector OP (where O is the sensor’s center, and P is the palm center) and the X axis. We have defined eight angle values to determine the position of the hand. Each angle forms an arc of 45 °, and thus these eight angles fit inside a circle. If this angle’s value falls within a given arc, it will then correspond to the position related to the arc. This angle is measured when a tap is detected outside the circle threshold. Figure 4.1 illustrates the arcs and the hand's position depending on the arc, and Algorithm 6 explains how the detection is achieved.

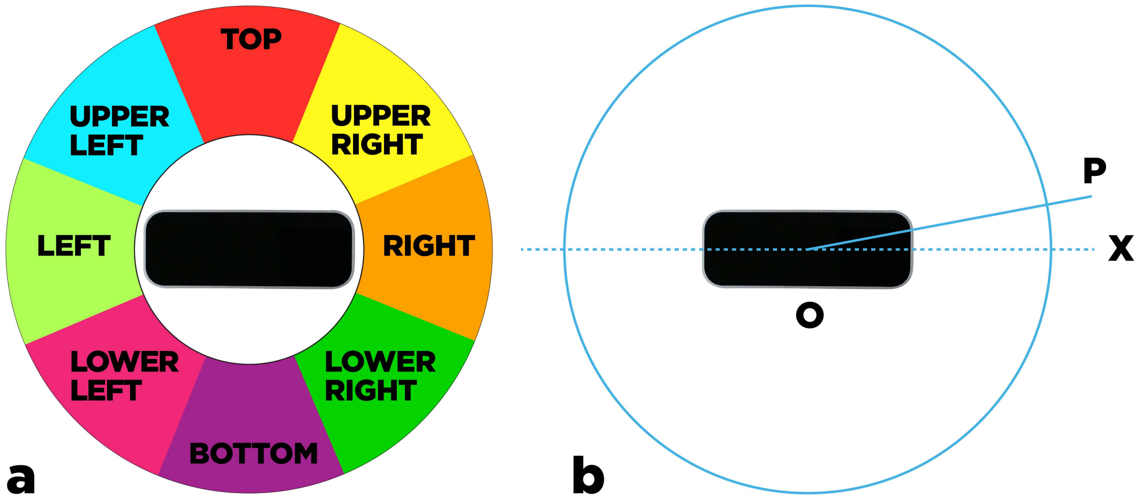


Figure 4.1: a shows the distribution of the arcs around the circle. b shows how to find the position by measuring the angle XOP, as long as the hand is outside the circle. Here, P stands for the palm center (position of the hand), O stands for the origin (position of the sensor), and X corresponds to a point on the X axis, positioned to the right of O.

Algorithm 6 Detection of a multi-click

```
procedure OnClick
    if ($\overline{OP}$ < radius) then
        return
    end if
    Determine angle ∠ POX // OX is the horizontal X axis
    Deduce position from ∠ POX
end procedure
```

4.5.2 Midair multi-click keyboard implementation

In our research, we suppose that the users will utilize the midair multi-click keyboard in the same way they do a physical QWERTY keyboard. We have used the same layout that was used in 3.6.2, and the fingers were assigned to the keys in the same way too. For example, the little finger of the left hand can tap the "Q, A or Z" keys. The possible keys on our keyboard are the twenty-six alphabet keys, in addition to the following four keys: semicolon (;), comma (,), period (.) and forward slash (/). The space key is also usable.

To input text using the midair multi-click keyboard, users place their hands about 20cm above the Leap Motion sensor and then perform midair taps as shown in Figure 4.2.

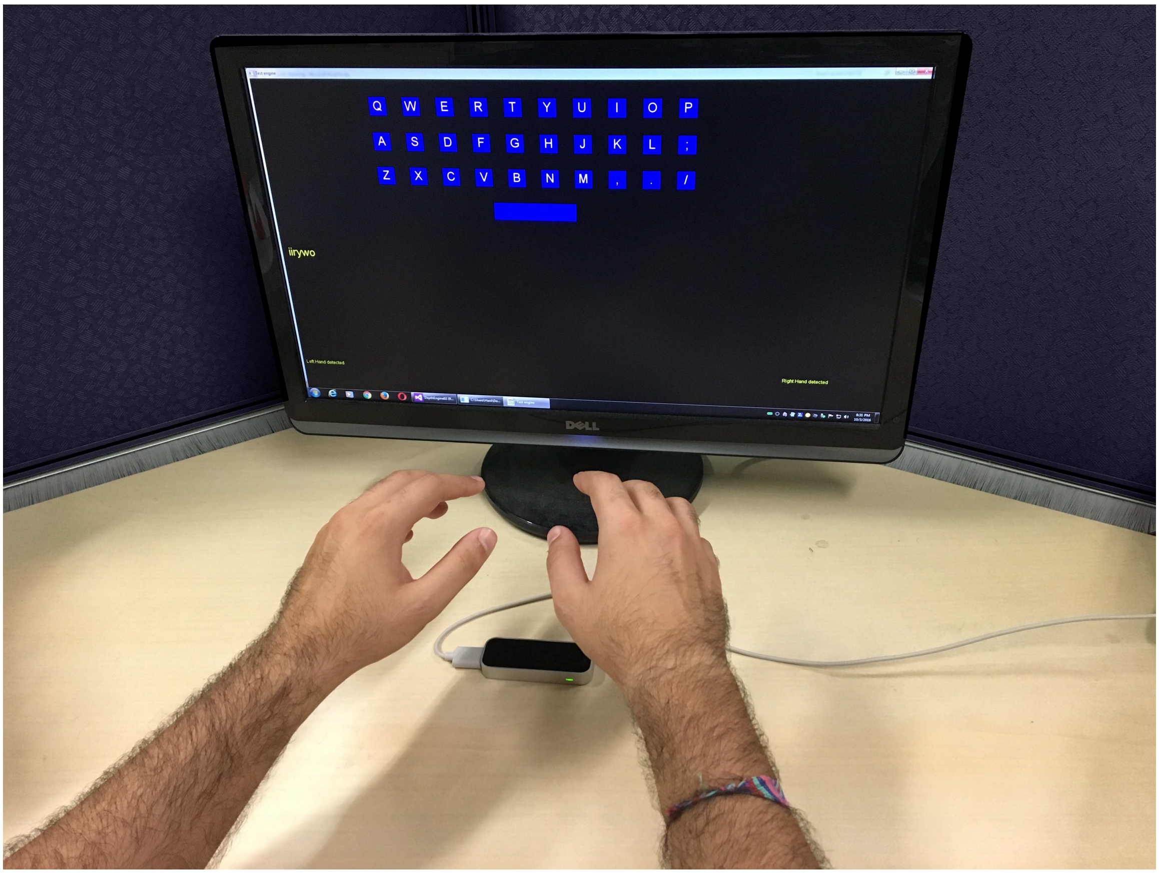


Figure 4.2: Midair multi-click keyboard prototype.

But the main advantage of our keyboard is that it can detect modifier keys such as “Shift”, “Control” or “Alt” in a multi-click fashion: users can tap and hold a modifier key with one hand, while inputting regular keys with the other hand, just as it is done when typing on a physical keyboard. To input a “modifier key - regular key” combination, we assume that the hand inputting the modifier key is outside the circle defined in 4.5.1. In this position, a Thumb-click-down is executed. Upon detection of a Click Down gesture, the position of the hand palm is then retrieved, and is used to determine the angle ∠ XOP. This angle is then used to determine the position of the Click Down regarding the predefined locations in the circle (Figure 4.1). The final step would be determining which hand performed the gestures. If the right hand executed the tap, then the following algorithm is applied:

- A Shift-click occurs when the thumb is continuously clicking in the right arc (Figure 4.1-a)
- A Control-click occurs when the thumb is continuously clicking in the lower right arc (Figure 4.1-a)
- An Alt-click occurs when the thumb is continuously clicking in the bottom arc (Figure 4.1-a)

Conversely, if the left hand performed the gestures, then the modifier click is detected in the following way:

- A Shift-click occurs when the thumb is continuously clicking in the left arc (Figure 4.1-a)
- A Control-click occurs when the thumb is continuously clicking in the lower left arc (Figure 4.1-a)
- An Alt-click occurs when the thumb is continuously clicking in the bottom arc (Figure 4.1-a)

To detect a regular key tap, we first consider that the hand is inside the circle defined in 4.5.1. Here, we use the same approach that was used in the keyboard from 3.5. Since it was determined that identifying the keyboard's row using the Palm Position Approach (3.7.3) was the most accurate, we have used that technique in the current midair keyboard. We summarize the multi-click midair keyboard as the following:

- If a thumb click was detected from either hand, a space key tap is generated, regardless of the position of the tap.
- If a middle finger, ring finger, or little finger click was detected, we deduce the row of the tap, then generate a click of the corresponding key.
- In the case of an index click, we deduce both the row and the column. We then generate the appropriate key tap.

To detect the row of a given tap, we determine the position of the hand relatively to the Leap Motion sensor, and use it in the following way:

- If the hand is right above the sensor, and inside a given margin, then we assume that the tap occurred in the middle row of the keyboard.

- If the hand moves away from the user's body, and beyond the above mentioned middle row margin, we determine that the finger tap occurred in the upper row.
- Conversely, if the hand moves closer to the user's body, and below the middle row margin, then we suppose that the finger tap took place in the lower row.

4.5.3 Possibility of one-handed multi-click detection

Since MultiX Click is based on Xpli Tap to detect the finger taps, it also uses its finger isolation technique (3.4.5). As a consequence, only one key can be detected at a given time. However, it is not impossible to detect a multi-click generated from one hand.

Our system detects a basic tap by separating it into two steps: a Click Down followed by a Click Up, as discussed in 2.5.2, and raises the respective events when they are detected. This makes it possible to detect the Click Down events of each finger individually, which can be used for a multi-click detection. As an example, we can consider the scenario where we need to detect a simultaneous click of the Ring and Index fingers from the left hand. The following steps can be used to distinguish this combination:

1. Detect the Click Down even of the Ring finger.
2. Detect the tap even of the Index.
3. If the Index tap was detected when the Ring finger was generating the Click Down event, then we consider that a multi-click was generated.
4. The Ring finger is then released (it then generates a Click Up event).

Due to the design of the tap detection of Xpli Tap, the following problems might occur:

- To be successfully detected, the Index finger has to go deeper than the Ring finger.
- The Ring finger has generated both the Click Down and Click Up events, and as a consequence, might generate a tap.

These limitations exist due to the design of Xpli Tap; however, circumventing them is feasible. For example, a time limit can be added to the Ring finger; once it crosses a predefined time threshold, it will not generate a tap anymore, but would only be used for a single-handed multi-click gesture. In our study, we did not implement this design, however, it is not impossible. The current limitations arise from the very nature of the keyboard where detecting multiple simultaneous keystrokes might generate an erroneous input. By removing the keyboard scenario from the equation, a single-handed multi-click gesture becomes realizable.

Another possibility is to detect multi-taps from a single hand, when that hand is in the multi-click area defined in 4.5.1 and illustrated in (Figure 4.1 - a). Whenever a hand is in this zone, the finger isolation algorithm will not be triggered; this makes detecting multiple fingers originating from a single hand possible. For example, if the Shift and the Ctrl keys are mapped respectively the thumb and index fingers, the right hand might input the combination "Shift + Ctrl" by simultaneously bending the thumb and index fingers. Furthermore, the left hand can input a single character such as "F", for a total combination of "Shift + Ctrl + F". This kind of multi-digits bending and simultaneous multi-tap detection will be explained in details in 4.6.

4.6 Air Piano Application

Our system can detect different forms of simultaneous multiple clicks, such as a Click Down in one hand mixed with a regular tap in the other (as used in the midair multi-click keyboard). Another possibility is detecting all the Click Down gestures that are occurring in one hand. As an application to this situation, we have created an air piano. An air piano is an ideal platform to test our multi-click detection system, especially that when pianists are playing the piano, they hit multiple keys and generate multiple notes at the same time. Our system can detect continuous Click Down motions, so we designed the air piano to keep on playing given notes as long as the related fingers are in Click Down position; this replicates sustained notes in a regular piano. Once fingers are not in Click Down position (after they perform the Click Up gestures), the respective notes stop playing. We have extended this to both hands, so users can play up to ten notes simultaneously.

In our version of the air piano, we have covered two octaves (14 white keys, 10 black keys); each hand would operate an octave. The left hand is assigned to the first octave; to right hand to the second. We first start by explaining the hand movement assumptions that we used to simulate playing the piano. We then explain how each hand operates its assigned octave separately (Figure 4.3).

A piano's keyboard has two rows: the lower row which consists of white keys, and the upper row which consists of black keys. When pianists are playing the piano, we suppose that they move their hands up (away from their torso) to reach the black keys, and down

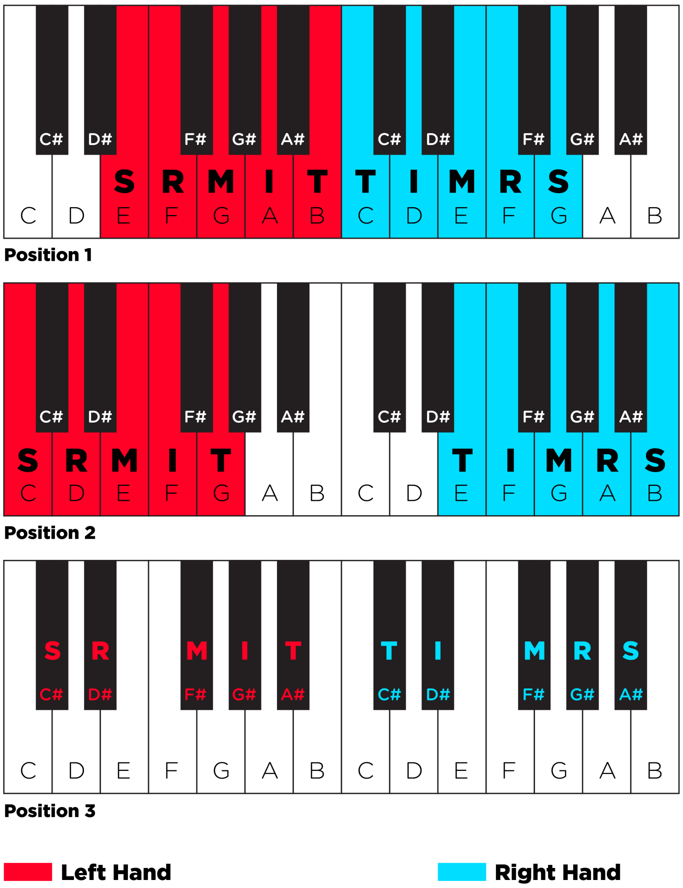

C# D# F# G# A# C# D# F# G# A#
S R M I T T I M R S
C D E F G A B C D E F G A B
Position 1
C# D# F# G# A# C# D# F# G# A#
S R M I T T I M R S
C D E F G A B C D E F G A B
Position 2
S R M I T T I M R S
C# D# F# G# A# C# D# F# G# A#
C D E F G A B C D E F G A B
Position 3
Left Hand
Right Hand

Figure 4.3: Keys detection in Air Piano. The letters S, R, M, I and T correspond respectively to the Small, Ring, Middle, Index and Thumb fingers. Position 1 shows how the left and right hands are assigned to the inner white keys. In Position 2, they are assigned to the outer white keys. Position 3 illustrates how the fingers hit the black keys.

(close to their torso to hit the white keys). To replicate this in our system, we supposed that the Leap Motion sensor sits in the center of our piano version (between the two octaves). We used the same angles model that we have previously utilized in the detection of the Shift, Control and Alt keys: whenever a left hand is in the upper left arc of the circle (Figure 4.1 - a), it will be considered that the users are trying to hit the black keys corresponding to that hand. If the right hand is in the upper right arc of the circle (Figure 4.1 - a), then we consider that the users want to reach the black keys related to the right hand. The next step is assigning fingers to the keys. In the case of black keys, each octave has five, so the fingers are directly assigned to the keys in the following manner: in the left hand, the little, ring, middle, index and thumb fingers are respectively assigned to the first octave's C#, D#, F#, G# and A# keys. The right hand's assignment's order will be reversed: the second octave's C#, D#, F#, G# and A# keys will respectively be assigned to the thumb, index, middle, ring and little fingers (Figure 4.3 - Position 3).

To hit the white keys, we assume that when the left and right hands are respectively in the left and right arcs (Figure 4.1 - a), the users want to hit the white keys. Unlike the black keys, which amount to five in each octave and can thus be directly assigned to the five fingers of each hand, there are seven white keys per octave, and hence a different approach is required to hit them. Therefore, we used the same approach that we had previously used in 4.5.1 to detect when to trigger multiple selection. We used a radius (Figure 4.1 - b) of 80 mm as a threshold. If the distance from the palm center to the Leap Motion sensor is less than this threshold, then we suppose that the users are trying to hit the inner notes of an octave; otherwise, they will be attempting to hit the outer notes. In the case of the left hand, the little, ring, middle, index and thumb fingers are respectively assigned to:

- E, F, G, A, B notes of the first octave, in the case of the *inner* notes (Figure 4.3 - Position 1)
- C, D, E, F, G notes of the first octave, in the case of the *outer* notes (Figure 4.3 - Position 2)

As for the right hand, the thumb, index, middle, ring and little fingers are respectively assigned to:

- C, D, E, F, G notes of the second octave, in the case of the *inner* notes (Figure 4.3 - Position 1)
- E, F, G, A, B notes of the second octave, in the case of the *outer* notes (Figure 4.3 - Position 2)

Our midair piano supports the simultaneous multi-click detection of MutiX Click, and thus allows the users to play multiple notes at the same time. It also produces an audio output of the corresponding played note to simulate a real piano operation. Visual feedback is also given to the users by changing the color of the key that was hit to green. Both the audio and visual feedbacks continue to operate as long as the corresponding key is hit (that is, as long as the assigned finger is in Click Down position).

The participants of the experiment also tried the air piano, and we have received multiple comments. Figure 4.4 shows the Midair Piano prototype in action.

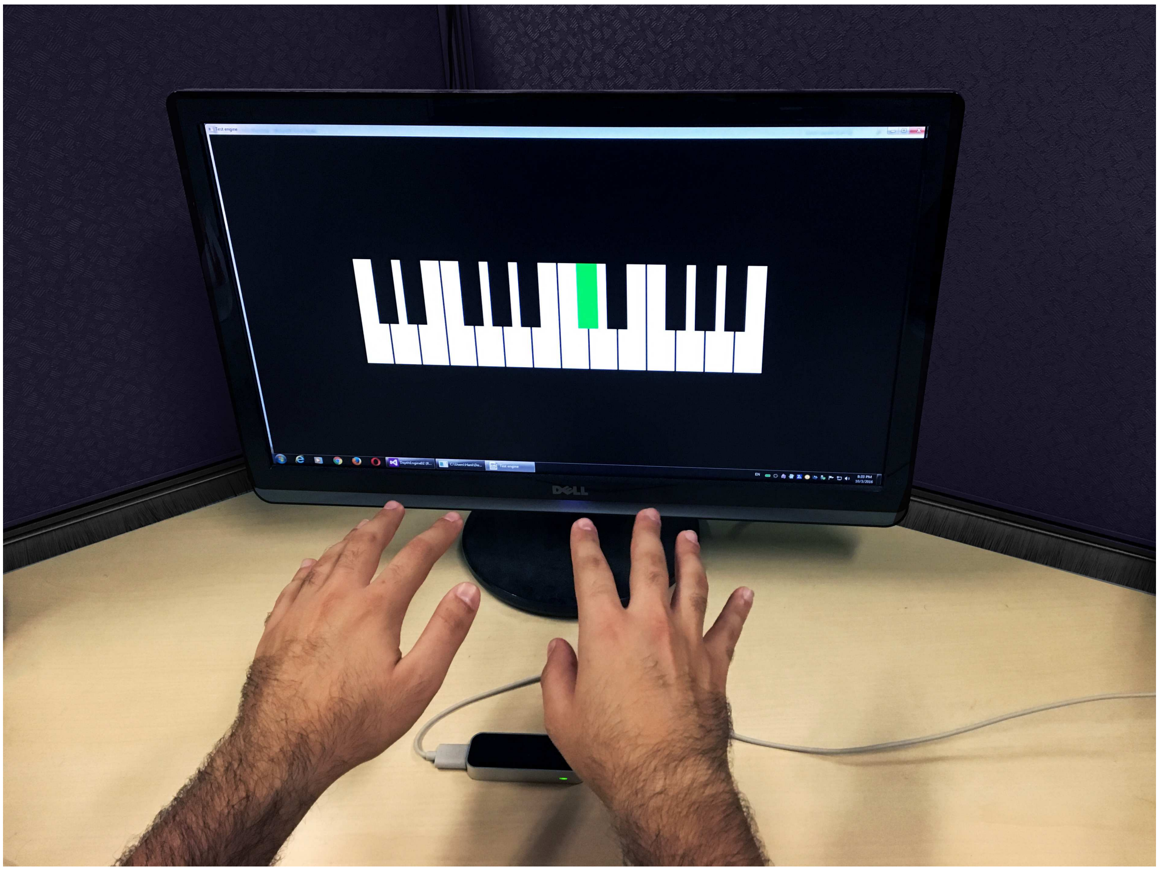


Figure 4.4: Midair Piano prototype. The active key's color is changed to green as a visual feedback to the user.

4.7 Evaluation

To evaluate MultiX Click, we conducted three different experiments. Each experiment is meant to evaluate a specific aspect of the method. The experiments are detailed here below in their respective subsections.

4.7.1 Experiment 1

In the first experiment, we wanted to evaluate the effect of multi-clicking on the input accuracy. To evaluate our system, 12 participants (all males) aged between 20 and 29 (average 23.75, S.D 2.42) were recruited; 9 amongst them were computer science / engineering students. All were right-handed. A brief introduction was given to the participants on how to carry out the experiment. They were asked to practice using the system for about 10 minutes. To perform the experiment, the participants placed their elbows on the desk, and their hands about 20 cm above the Leap Motion sensor, as can be seen in Figure 3.9. In one variation of the experiment, the participants were asked to input all thirty-one keys while using the three modifier keys Shift, Control, and Alt. In this case, they have to input a combination of a modifier key and a regular key (for example, “Shift A” or “Control P”). To input this kind of combination, they used one hand for inputting the modifier key, and the other for the regular key. When a key from the left hand side of the keyboard was being inputted (such as R or D), the modifier key would be inputted with the right hand. Conversely, when a key from the right hand side of the keyboard was being used (such as N or U), then the left hand would input the modifier key. Just like with a physical keyboard, the modifier key should be held down, then the regular key tapped. To translate this into the midair keyboard, a modifier key is first activated by performing a Click Down gesture, and holding it (the finger does not go up). Then, with the opposite hand, a key is inputted in the normal way (Click Down followed by a Click Up). After that, the finger inputting the modifier key can be released (Click Up). In the experiment, the participants start by using the right hand to input the modifier keys by using a Click Down gesture. While keeping the Click Down gesture executed, they then, with their left hand, input a key. Each “modifier key + regular key” combination was entered five times (Shift + Q, Shift + Q, Shift + Q, Shift + Q, Shift + Q, Ctrl + Q, Ctrl + Q, Ctrl + Q, Ctrl + Q, Ctrl + Q, Alt + Q, Alt + Q, Alt + Q, Alt + Q, Alt + Q) They started by inputting the keys corresponding to the upper row of the keyboard (Q, W, E, R, T), then moved on to the middle row (A, S, D, F, G). Finally, they continued with the bottom row (Z, X, C, V, B). All those keys are inputted in combination with modifier keys. Next, they switch hands: they input modifier keys with their left hand, while at the same time inputting regular keys with their right hand. They started with the upper row (Y, U, I, O, P), middle row (H, J, K, L, ;) and bottom row (N, M, comma, period, forward slash). Finally, they entered the space key with their preferred hand, while simultaneously entering the modifier keys with the opposite hand.

The other variation of the experiment is similar to the previous one. The only exception is that participants would not enter any modifier key. In this task, the participants were asked to input all thirty-one keys, fifteen times each (QQQ..., WWW..., EEE, ...). They started with their left hand, and inputted the keys corresponding to the upper row of the keyboard (Q, W, E, R, T). They then inputted the keys from the middle row (A, S, D, F, G). Finally, they continued with the bottom row (Z, X, C, V, B). They then repeated the same task using their right hand, where they started with the upper row (Y, U, I, O, P), middle row (H, J, K, L, ;) and bottom row (N, M, comma, period, forward slash). Finally, they entered the "space" key fifteen times. The participants were allowed to take a break whenever they felt tired, which was almost once after each row (six times per experiment). To prevent the order effect, the order of the experiment variations was changed for every participant.

4.7.2 Questionnaire 1

After completing the experiments, the participants were asked to fill out the following questionnaire, to which they could respond on a five point Likert scale (-2 = Strongly negative answer, 2 = Strongly positive answer):

1. To what extent does multiple selection's usage feel natural?
2. To what extent is multiple selection's usage easy?
3. To what extent did multiple selection operate as you thought?

The participants were also offered the opportunity to enter comments, if any, freely.

4.7.3 Results Of Experiment 1

Each participant inputted a key 15 times, while at the same time inputting a modifier key with the opposite hand. As a result, each key involved 2 separate inputs, and thus the total inputs per key were 30. This was repeated for all 31 keys of the keyboard, which sums up to 30 * 31 = 960 in the multi-click variation. In the other variation, each participant executed 15 * 31 = 465 inputs, for a total of 1,425 input per participant. All in all, the 12 participants performed 17,100 inputs. We have analyzed the input accuracy for each experiment variation. Figure 4.5 and Figure 4.6 show the results per key for the regular input variation and modifier key variation respectively.

4.7.4 Results Of Questionnaire 1

The results of the questionnaire are shown in Table 4.1

The result of Question 1 shows that the participants agreed that the usage of multiple selection is natural, whereas the result of Question 2 indicates that they marginally agreed that it was easy. Finally result of Question 3 indicates that the participants marginally agreed that multiple selection operated as they thought.

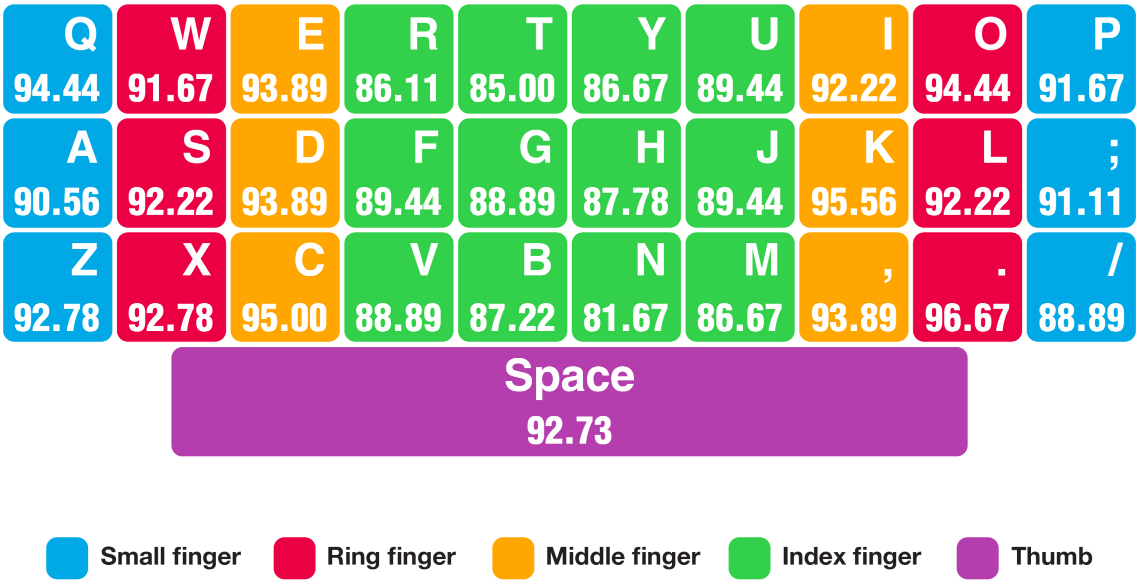


Figure 4.5: Success rate (%) by key without using Multi-click.

Table 4.1: Questionnaire 1 results.

Question	Mean	S.D
Question 1	0.83	0.72
Question 2	0.58	0.79
Question 3	0.50	0.67

4.7.5 Experiment 2

In this experiment, we wish to evaluate the effect of multi-clicking on the input speed in our air keyboard. 12 participants (all males) aged between 20 and 26 (average 23.42, S.D 1.62) were recruited; 5 amongst them were computer science / engineering students. All were right-handed. A brief introduction was given to the participants on how to carry out the experiment. They were asked to practice using the system for about 10 minutes. The participants were asked to input the following 5 sentences:

1. “1 Year Has 7 Months With 31 Days”
2. “There are 102248 arachnida species on Earth”
3. “HTML5 is better than HTML4”
4. “Mount Tsukuba Is 877 Meters High”
5. “Star Wars Was Released On May 25th 1977”

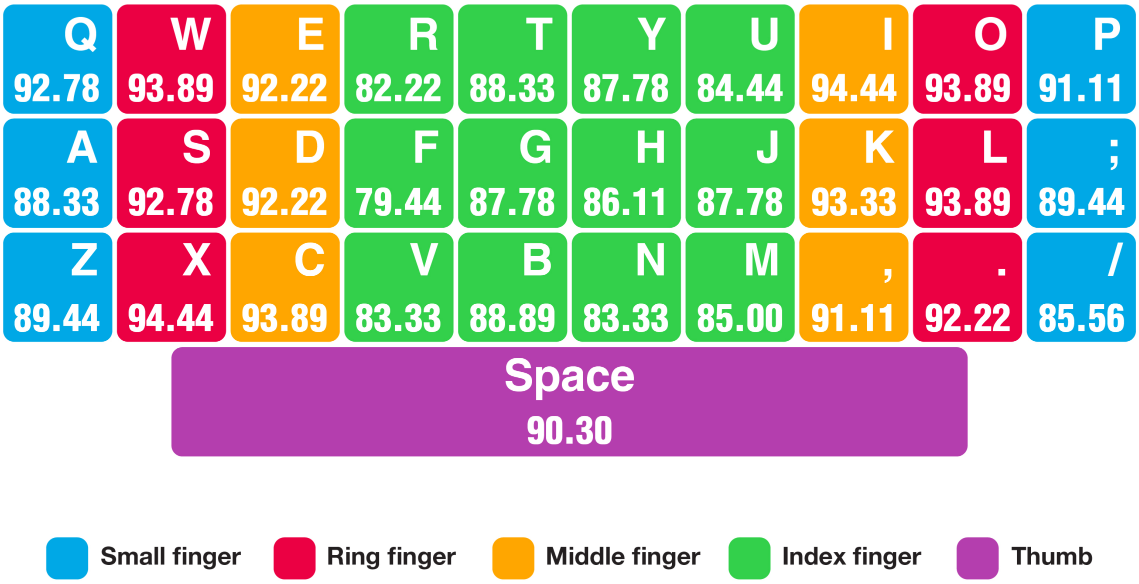


Figure 4.6: Success rate (%) by key while using Multi-click.

The participants had to respect the case of the letters as they appeared in the sentences. They were asked to input each sentence in two different ways. In the first one, they were able to input capital letters by clicking down on the Shift key with one hand, and selecting the appropriate key with the other hand. Since our keyboard does not have keys to input numerical values, we mapped the numbers to a combination of Control button and the keys from the middle row (a, s, d, f, g, h, j, k, l, semi-colon). Ctrl + a would generate the number 1, Ctrl + s would generate the number 2, and so on. Control + semi-colon would generate the number 0. In the second method, we added a “Capital Lock” and a “Number Lock” buttons on the left and right side of the keyboard respectively. Those two buttons can then be activated and deactivated in the same way as the modifier keys:

- The Capital Lock button is activated/deactivated when a thumb click occurs in the left arc (Figure 4.1-a)
- The Number Lock button is activated/deactivated if a thumb click is detected in the right arc (Figure 4.1-a)

Once the Capital Lock key is activated, the inputted text following that will be in upper case. To return back to lower case, it has to be deactivated. The Number Lock key operates in the same way. As with the previous method, the keys from the middle row (a, s, d, f, g, h, j, k, l, semi-colon) are mapped to the numbers 1, 2, 3, 4, 5, 6, 7, 8, 9, 0. To prevent the order effect, the order of the sentences and that of the input method were chosen randomly for each participant.

4.7.6 Results Of Experiment 2

We have measured the average time in seconds to complete inputting the sentence in both input methods. Furthermore, we analyzed the collected data using a paired t-test (α = 0.05). Table 4.2 shows the results.

Table 4.2: Results of Experiment 2. Single-click refers to the method where the Capital and Number lock keys were used. Multi-click refers to the input method where the Shift and Control keys were used to input capital letters and numbers respectively. Diff shows the difference of speed between the two methods.

Sentence	Single-click	Multi-click	T-test (p)	Difference
1	52.21s (S.D. 14.22)	38.65s (S.D. 15.10)	0.013	26%
2	43.30s (S.D. 12.06)	43.07s (S.D. 14.74)	0.967	1%
3	31.37s (S.D. 8.50)	33.45s (S.D. 10.06)	0.590	6%
4	44.20s (S.D. 12.49)	34.00s (S.D. 9.84)	0.038	23%
5	57.10s (S.D. 15.61)	43.02s (S.D. 11.16)	0.019	25%

4.7.7 Experiment 3

In this experiment, we wanted to evaluate the multi-click detection in Air Piano. For this purpose, 10 participants (all males) aged between 22 and 26 (average 23.5, S.D 1.08) were recruited; 8 amongst them were computer science students. 9 were right-handed. A brief introduction was given to the participants on how to carry out the experiment. They were asked to practice using the system for about 5 minutes. In this experiment, the participants were asked to input 3 different piano chords, 10 times with each hand. To prevent the learning effect, the hand, and the chord that was executed by that hand were selected at random each time. Piano chords involve hitting multiple notes at the same time. The following chords were used:

- C Major, using C, E and G notes
- C sus 2, using C, D and G notes
- C sus 4, using C, F and G notes

As explained in 4.6, the notes C, D, E, F and G are respectively mapped to the little, ring, middle, index and thumb fingers of the left hand, and to the thumb, index, middle, ring and little fingers of the right hand. Therefore, the fingers used to perform the chords are different for each hand. Table 4.3 and Table 4.4 show the fingers used respectively in the left and right hands to execute the chords.

Table 4.3: Fingers used in the left hand to execute the chords. The corresponding note is shown in parenthesis.

Chord	Finger 1	Finger 2	Finger 3
C Major	Little (C)	Middle (E)	Thumb (G)
C sus 2	Little (C)	Ring (D)	Thumb (G)
C sus 4	Little (C)	Index (F)	Thumb (G)

Table 4.4: Fingers used in the right hand to execute the chords. The corresponding note is shown in parenthesis.

Chord	Finger 1	Finger 2	Finger 3
C Major	Thumb (C)	Middle (E)	Little (G)
C sus 2	Thumb (C)	Index (D)	Little (G)
C sus 4	Thumb (C)	Ring (F)	Little (G)

4.7.8 Results Of Experiment 3

We evaluated the chords detection in Air Piano. Each chord is inputted with specific fingers. For a positive detection, only the required fingers had to be detected. Any other condition (less than the required fingers were detected, more than the required fingers were detected, no detection at all) was considered negative and was rejected. We duly note that all chords executions were detected, and thus the results reflect whether the right or wrong combination of fingers was detected. Table 4.5 shows the results of the chords detection rate.

Table 4.5: Results of the average chords detection rate per hand (the standard deviation is shown in parenthesis).

Hand	C Major	C sus 2	C sus 4
Left	44% (3.13)	83% (1.77)	83% (1.70)
Right	34% (4.30)	79% (1.79)	91% (0.88)

4.7.9 Questionnaire 2

After completing the experiment, the participants were asked to fill out the following questionnaire, to which they could respond on a five point Likert scale (-2 = Strongly negative answer, 2 = Strongly positive answer):

1. To what extent is the C Major chord input easy?
2. To what extent is the C sus 2 chord input easy?
3. To what extent is the C sus 4 chord input easy?

The participants were also offered the opportunity to enter comments, if any, freely.

4.7.10 Results Of Questionnaire 2

The results of the questionnaire are shown in Table 4.6

Table 4.6: Questionnaire 2 results.

Question	Mean	S.D
Question 1	-0.40	1.43
Question 2	1.50	0.53
Question 3	0.80	0.92

4.7.11 Air Piano Impressions

We have received many comments on the Air Piano application, mostly positive. One participant commented that he learned piano for six years and was very satisfied with the air piano. Another stated that he was able to play a section of the “Do-Re-Mi song” after some practice. Other comments included “the air piano was very natural to use”, “the air piano was fun”, and “I think people can use the air piano to practice on the train”. Others commented that “changing octaves was a bit difficult” and “unintentional keys were detected some times”.

4.8 Discussion

In this section, we discuss the results of the three experiments and the possibilities of our system. We start by explaining the target of the midair multi-click recognition rate. We have defined the following three levels for the recognition rate:

- Level 0: 90% accuracy.

- Level 1: 99% accuracy.
- Level 2: 99.9% accuracy.

We have also defined the following 3 levels for input speed:

- Level 0: 8 strokes per minute - we can input in any way.
- Level 1: 40 strokes per minute - we can input with some stress.
- Level 2: 200 strokes per minute - we can input with no stress.

The target recognition rate and the target speed will depend on applications. For the Midair Keyboard application, our initial goals were Level 1 for accuracy and Level 1 for speed. For the Air Piano application, our initial goals were Level 0 for accuracy and Level 1 for speed. As can be seen from the results of Experiments 1 and 2, the Midair Keyboard goals have almost been satisfied.

Experiment 1 shows that we can type letters with a reasonable recognition rate. If we use multi-selection, we can also type with a reasonable recognition rate, i.e., multi-click does not lower the recognition rate drastically.

Experiment 2 shows that we can type regular English sentences with a certain speed. Comparing single-click and multi-click, multi-click seems to provide us with shorter input time when the words start with a capital letter. On the other hand, when consecutive numbers are appearing, multi-click does not shorten the input time.

Experiment 3 shows the detection and recognition rates of multi-taps. Previous research [48], [49] shows that human fingers do not move independently, and that motions were produced in other fingers even if the subjects were asked to move just one finger. This was especially present in the case of the C Major chord, due to the specific combination of the required fingers. Therefore, the recognition rate of the C Major chord dropped. This problem was less obvious for the C sus 2 and C sus 4 chords which had higher recognition rates. Our system detected all the inputs in Experiment 3 (100% detection rate), however, due to the constraints of the hand anatomy, some rates dropped. But we can say that the above accuracy goal for Air Piano has been satisfied. Moreover, since we can play the Air Piano in real time, we can say Level 1 speed has also been realized.

By introducing finger tapping, and using different fingers, we can input gesture sequences in a smoother way, which is a great advantage compared to one-shot finger shape gesture. By explicitly detecting finger taps, we were able to have a great control over the interaction. We were able to precisely define the beginning and the end of gestures by using clicks. Our system successfully detects single taps, “hybrid multi-taps” using both hands (two taps, one coming from each hand, are detected simultaneously) and “pure multi-taps” (multiple fingers clicked in one hand). Moreover, in the latter case, the identity of the fingers is also detected and not just the number of multi-clicking fingers (i.e. a multi-click with the index and thumb is different than a multi-click with the index and ring fingers, even though in both cases two fingers are clicked). Identifying the fingers that tap in a single hand expands the vocabulary of the interaction in that given hand. Finally, combining single taps, hybrid multi-taps and pure multi-taps, such as mixing pure multi-clicks in both hands, gives rise to multiple possibilities and thus notably increases the overall vocabulary of midair taps.

4.9 Other Possible Applications

A potential application for our system is a midair guitar player. Since our system can distinguish multi-taps this can be used to estimate the chords that a player is trying to play.

Typically, the ability to use multi-clicking enriches the gesture interaction vocabulary. This increase of vocabulary is very flexible too: the combination of multi-clicks can be used to differentiate gestures. For example, simultaneous clicking of different fingers (in the same hand) can generate different gestures, as has been shown in Experiment 3. This can be mixed with regular taps from the other hand. A possible application to this scenario is a secure input system for passwords or PINs. For example, let’s suppose that a user’s PIN is 123. Each of the digits 1, 2 and 3 will be inputted using a regular tap in one hand, while simultaneously performing different multi-clicks in the other hand. Therefore, the PIN is not just 123 anymore, but a combination of numbers in one hand and multi-tapping with different fingers in the other hand, which makes its input more secure and decreases the chances of being guessed. Another possible application to multi-tapping is 3D multi-touch gestures, the same way multi-touch is used in smartphones or tablets by using multiple fingers. For example, we can click with the index and middle fingers simultaneously, then swipe the hand to trigger a scroll depending on the swipe direction. Moreover, a 2-finger tap can be used to generate a right-click action, and a pinch with thumb and index fingers can be used to zoom in or out of a picture.

4.10 Summary

In this chapter, we presented MultiX Click, a new midair multi-click detection method. We used Xpli Tap which was introduced in Chapter 3 as our starting point, and added the detection of simultaneous multi-taps that are executed using multiple fingers. Xpli Tap isolates the tapping finger for greater accuracy; we have introduced a technique that distinguishes multiple zones which allows to mix single and multi-tapping simultaneously. This allowed the detection of concurrent taps occurring from both hands: in one hand, a Click Down was performed, and a regular tap was executed from the other hand. To show the advantage of this approach, we implemented a midair keyboard which allows the usage of the Shift, Control and Alt modifier keys by combining them with regular keys. Users can utilize these modifier keys by clicking down on them; they can use a normal click on any other key, as if using a physical keyboard. Furthermore, we have detected the multi-taps that are performed in one hand only. In this case, various fingers are bent simultaneously, and their taps are detected concurrently. A midair piano using this approach was implemented where users can play single notes or chords. We have conducted experiments that have shown that MultiX Click is a reliable method, and we have reached the original targets that we had designated as acceptable for our system. Besides detecting multi-taps, MultiX Click allows the identification of every finger. By distinguishing the number of tapping fingers and their identity, the interaction vocabulary can be augmented.

Chapter 5

Conclusion

The advancement of depth sensors and processing power have promoted the usage of natural interfaces using the body as the only input device. Unlike worn sensors, a depth camera does not need any wired connectors or donned devices in order to detect gestures. And while RGB cameras can and have been used to distinguish hand gestures, they have some limitations such as sensitivity to light and can only read images in 2D, thus reducing their accuracy. Depth cameras eliminate those problems by using infrared light and detecting a depth map which allows the measure of the distance between objects and the camera.

Different gestures can be detected using depth cameras, and a particularly interesting one is the tapping gesture. Tapping gestures are familiar to humans, as they are use to performing it on keyboards, mice and touchscreens. Moreover, tapping gestures are a good candidate for selection mechanisms which are used to distinguish multiple elements in a user interface. Ideally, a selection mechanism should not only have the ability to differentiate between different user interface elements, but should also allow the users to explicitly start and end a selection action, and cancel it if necessary. These particular points have not been covered in depth in the existing literature.

5.1 Contributions

Our research work has provided valuable methods to solve the limitations of the existing techniques in selection mechanisms and tap detection. We have detected a tapping gesture by extending the thumb and middle finger to generate a plane which was used to measure the angle of the index while executing a tap. Once the angle crossed a defined threshold, a tap was generated. This technique was then improved by using the normal vector of the palm to detect the angle, which freed the thumb and middle fingers from being always extended. Furthermore, this allowed a tap to be detected from every finger of the hand. Moreover, a tapping gesture was split into a Click Down and a Click Up. A full click was hereby defined as a Click Down followed by a Click Up. In the case of a user interface elements, an additional rule was introduced to detect a click. This rule stipulated that the Click Down and Click Up actions should be performed over the same target. This allowed the cancellation of the click even though it may have already started with a Click Down by simply moving the hand away from the target during a Click Up operation. Splitting the tap into two separate gestures introduced the ability to explicitly define the start and end of the gesture. A tap was therefore not limited by any time constraints. This allowed us to use the latter to our advantage; we were able to detect a Long Click as well as Double Click gestures by defining some time thresholds. Furthermore, we were able to detect those two gestures as well as a regular tap depending on the time threshold. Using our technique, we have implemented a prototype which allowed users to operate a photo viewer with midair finger taps. A thumb click would change the function from dragging to zooming to rotating. Then an index tap will allow the users to execute the selected function. During evaluation, our tap detection algorithm proved to be reliable.

We also proposed an algorithm to explicitly detect midair finger taps. First, we created a relaxed neutral posture by introducing a calibration step. Then we were able to extract the exact position of an executed tap by analyzing the fingertip position during the tapping action. Moreover, we were able to isolate the tapping finger when multiple fingers moved concurrently; this has led to an accurate detection of the gesture. Since we could detect the exact position of a tap, we used it to augment the interaction vocabulary by assigning different actions to a single finger depending on the location of the tap. Furthermore, we detected three different shapes for the tapping finger. By using these shapes along with a tapping action, we were also able to enrich the gestures vocabulary. A midair QWERTY keyboard prototype was created as an application to our algorithm. The results of the evaluation experiments have shown its efficiency.

Finally, we proposed a technique to detect multiple taps generated simultaneously from different fingers. We introduced an approach to be able to mix multi-taps with regular taps by defining a zone for each. To show the feasibility of this technique, we implemented a midair keyboard using the Shift, Control and Alt modifier keys. The users were able to simultaneously input a modifier key in one hand and a regular key with the other. Moreover, we were able to distinguish multiple taps coming from different fingers in a single hand. This made it possible to detect concurrent taps similar to multi-touch gestures in touchscreens. We implemented a midair piano where users could play one or simultaneous multiple notes such as chords depending on the number of fingers that they bend. Both of these approaches have noticeably increased the gestures vocabulary. The evaluation showed that our technique is reliable.

Our proposed methods in midair finger tap detection can overcome some of the limitations of the existing techniques. A possible application of our work is incorporating these methods in portable devices such as smartphones and smartwatches. Nowadays, smartphones have become powerful enough to allow users to send emails and conduct business using them, and smartwatches may in the future be as powerful. However, interacting with those devices is still limited, mainly because of the small size of their screens. The possible interactions can be increased if those devices are equipped with an adequate sensor; users will be able to use them without being restricted by the limited size of their screens. For example, a smartphone can be laid down on a horizontal surface, and then the user can type a full email just by placing his or her hands above the screen and tapping in midair.

Computers have evolved from the terminal form (small monitor and keyboard) to new models including smartphones, large screens and smartwatches. Along with these new computers comes the need for new ways of interactions, besides mice and keyboards. Another contribution of our research is that it moves a step closer towards liberating users from the restrictions of the physical interfaces. In those interfaces, users have to position their hands around physical objects, and are forced to have their hands take the shape of the interfaces, but our midair system can detect the gestures in 3D, making it possible for interfaces to take the shape of the human hands. As a consequence, the interactions may become more natural by moving away from machines and closer to the users, and repetitive stress injuries may be reduced.

5.2 Future Work

Our study showed that midair finger tapping can be a promising technique to interact with computers. In the future, we would like to optimize the interactions and reduce the error rates; we may include machine learning and dictionaries to make the detection and keyboard more reliable. Furthermore, as Virtual Reality and Augmented Reality are becoming increasingly popular recently, we would like to investigate the integration of our system with these technologies, as they present users with virtual spaces that can be good candidates for midair interactions. Moreover, we would like to study the fatigue resulting from these interfaces and its effect on applications.

Bibliography

[1] S.-H. Lee, M.-K. Sohn, D.-J. Kim, B. Kim, and H. Kim, "Smart tv interaction system using face and hand gesture recognition," in *Consumer Electronics (ICCE), 2013 IEEE International Conference on.* IEEE, 2013, pp. 173–174.

[2] M. Takahashi, M. Fujii, M. Naemura, and S. Satoh, "Human gesture recognition system for tv viewing using time-of-flight camera," *Multimedia tools and applications*, vol. 62, no. 3, pp. 761–783, 2013.

[3] S. Cheema and J. J. LaViola, "Wizard of wii: toward understanding player experience in first person games with 3d gestures," in *Proceedings of the 6th International Conference on Foundations of Digital Games.* ACM, 2011, pp. 265–267.

[4] J. J. LaViola Jr, "Context aware 3d gesture recognition for games and virtual reality," in *ACM SIGGRAPH 2015 Courses.* ACM, 2015, p. 10.

[5] A. Bigdelou, L. Schwarz, T. Benz, and N. Navab, "A flexible platform for developing context-aware 3d gesture-based interfaces," in *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces.* ACM, 2012, pp. 335–336.

[6] L. Schwarz, A. Bigdelou, and N. Navab, "Learning gestures for customizable human-computer interaction in the operating room," *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2011*, pp. 129–136, 2011.

[7] P. Garg, N. Aggarwal, and S. Sofat, "Vision based hand gesture recognition," *Engineering and Technology*, vol. 3, no. 1, pp. 821–826, 2009.

[8] J. P. Wachs, M. Kölsch, H. Stern, and Y. Edan, "Vision-based hand-gesture applications," *Communications of the ACM*, vol. 54, no. 2, pp. 60–71, Feb. 2011.

[9] S. Chu and J. Tanaka, “Hand gesture for taking self portrait,” in *Human-Computer Interaction. Interaction Techniques and Environments*, 2011, pp. 238–247.

[10] J.-C. Lévesque, D. Laurendeau, and M. Mokhtari, “An asymmetric bimanual gestural interface for immersive virtual environments,” in *Virtual Augmented and Mixed Reality. Designing and Developing Augmented and Virtual Environments*, 2013, pp. 192–201.

[11] J. Accot and S. Zhai, “More than dotting the i’s — foundations for crossing-based interfaces,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2002, pp. 73–80.

[12] T. Nakamura, S. Takahashi, and J. Tanaka, “Double-crossing: A new interaction technique for hand gesture interfaces,” in *Computer-Human Interaction*, 2008, pp. 292–300.

[13] H. Kenn, F. V. Megen, and R. Sugar, “A glove-based gesture interface for wearable computing applications,” in *4th International Forum on Applied Wearable Computing 2007*, 2007, pp. 1–10.

[14] P. Kumar, J. Verma, and S. Prasad, “Hand data glove: a wearable real-time device for human-computer interaction,” *International Journal of Advanced Science and Technology*, vol. 43, 2012.

[15] C. v. d. M. Jochen Triesch, “Robotic gesture recognition by cue combination,” in *Informatik ’98*, 1998, pp. 223–232.

[16] Y. Boussemart, F. Rioux, F. Rudzicz, M. Wozniewski, and J. R. Cooperstock, “A framework for 3d visualisation and manipulation in an immersive space using an untethered bimanual gestural interface,” in *VRST ’04 Proceedings of the ACM symposium on Virtual reality software and technology*, 2004, pp. 162–165.

[17] J. Kovac, P. Peer, and F. Solina, *Human skin color clustering for face detection.* IEEE, 2003, vol. 2.

[18] S. Chu and J. Tanaka, “Design of a motion-based gestural menu-selection interface for a self-portrait camera,” *Personal and Ubiquitous Computing*, vol. 19, no. 2, pp. 415–424, 2015.

[19] S. Baker and I. Matthews, “Lucas-kanade 20 years on: A unifying framework,” *International journal of computer vision*, vol. 56, no. 3, pp. 221–255, 2004.

[20] H.-S. Yeo, B.-G. Lee, and H. Lim, "Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware," *Multimedia Tools and Applications*, vol. 74, no. 8, pp. 2687–2715, 2015.

[21] Kinect. [Online]. Available: https://developer.microsoft.com/en-us/windows/kinect

[22] F. Dominio, M. Donadeo, G. Marin, P. Zanuttigh, and G. M. Cortelazzo, "Hand gesture recognition with depth data," in *ARTEMIS '13 Proceedings of the 4th ACM/IEEE international workshop on Analysis and retrieval of tracked events and motion in imagery stream*, 2013, pp. 9–16.

[23] K. Lai, J. Konrad, and P. Ishwar, "A gesture-driven computer interface using kinect," in *Image Analysis and Interpretation (SSIAI), 2012 IEEE Southwest Symposium on.* IEEE, 2012, pp. 185–188.

[24] K. K. Biswas and S. K. Basu, "Gesture recognition using microsoft kinect®," in *Automation, Robotics and Applications (ICARA), 2011 5th International Conference on.* IEEE, 2011, pp. 100–103.

[25] U. Lee and J. Tanaka, "Finger identification and hand gesture recognition techniques for natural user interface," in *APCHI '13 Proceedings of the 11th Asia Pacific Conference on Computer Human Interaction*, 2013, pp. 274–279.

[26] J. Segen and S. Kumar, "Look ma, no mouse!" in *Communications of the ACM*, 2000, pp. 102–109.

[27] F. van de Camp, A. Schick, and R. Stiefelhagen, "How to click in mid-air," in *Distributed, Ambient, and Pervasive Interactions: First International Conference, DAPI 2013. Proceedings*, 2013, pp. 78–86.

[28] G. Apitz and F. Guimbretière, "Crossy: A crossing-based drawing application," in *UIST '04 Proceedings of the 17th annual ACM symposium on User interface software and technology*, 2004, pp. 3–12.

[29] M. Liu, M. Nancel, and D. Vogel, "Gunslinger: Subtle arms-down mid-air interaction," in *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, 2015, pp. 63–71.

[30] A. Kulshreshth and J. Joseph J. LaViola, "Exploring the usefulness of finger-based 3d gesture menu selection," in *CHI '14 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014, pp. 1093–1102.

[31] J. Joseph J. LaViola, "An introduction to 3d gestural interfaces," in *ACM SIGGRAPH 2014 Courses*, 2014, pp. 25:1–25:42.

[32] Depthsense 325. [Online]. Available: http://www.softkinetic.com/Store/tabid/579/ProductID/6/language/en-US/Default.aspx

[33] Allegro. [Online]. Available: http://liballeg.org/

[34] Creative senz 3d. [Online]. Available: http://us.creative.com/p/web-cameras/creative-senz3d

[35] Observer design pattern. [Online]. Available: https://msdn.microsoft.com/en-us/library/ee850490(v=vs.110).aspx

[36] Sfml. [Online]. Available: http://www.sfml-dev.org

[37] F. Guimbretière and C. Nguyen, "Bimanual marking menu for near surface interactions," in *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, 2012, pp. 825–828.

[38] J. F. Kelley, "An empirical methodology for writing user-friendly natural language computer applications," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM, 1983, pp. 193–196.

[39] K.-Y. Chen, K. Lyons, S. White, and S. Patel, "utrack: 3d input using two magnetic sensors," in *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM, 2013, pp. 237–244.

[40] S.-Y. Lin, C.-K. Shie, S.-C. Chen, and Y.-P. Hung, "Airtouch panel: a re-anchorable virtual touch panel," in *Proceedings of the 21st ACM international conference on Multimedia*. ACM, 2013, pp. 625–628.

[41] H. Bai, G. A. Lee, and M. Billinghurst, "Freeze view touch and finger gesture based interaction methods for handheld augmented reality interfaces," in *Proceedings of the*

27th Conference on Image and Vision Computing New Zealand. ACM, 2012, pp. 126–131.

[42] W. Hürst and C. Van Wezel, "Gesture-based interaction via finger tracking for mobile augmented reality," *Multimedia Tools and Applications*, vol. 62, no. 1, pp. 233–258, 2013.

[43] M. Baldauf, S. Zambanini, P. Fröhlich, and P. Reichl, "Markerless visual fingertip detection for natural mobile device interaction," in *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services.* ACM, 2011, pp. 539–544.

[44] A. H. Lam and W. J. Li, "Mids: Gui and tui in mid-air using mems sensors," in *Proc. ICCA*, 2002, pp. 1218–1222.

[45] C. Harrison, H. Benko, and A. D. Wilson, "Omnitouch: wearable multitouch interaction everywhere," in *Proceedings of the 24th annual ACM symposium on User interface software and technology.* ACM, 2011, pp. 441–450.

[46] A. Sugiura, M. Toyoura, and X. Mao, "A natural click interface for ar systems with a single camera," in *Proceedings of Graphics Interface 2014*, 2014, pp. 67–75. [Online]. Available: http://dl.acm.org/citation.cfm?id=2619648.2619660

[47] V. M. Zatsiorsky, Z.-M. Li, and M. L. Latash, "Coordinated force production in multi-finger tasks: finger interaction and neural network modeling," *Biological Cybernetics*, vol. 79, no. 2, pp. 139–150, 1998. [Online]. Available: http://dx.doi.org/10.1007/s004220050466

[48] Z.-M. Li, S. Dun, D. A. Harkness, and T. L. Brininger, "Motion enslaving among multiple fingers of the human hand," in *Motor Control*, 2004, pp. 1–15.

[49] C. Häger-Ross and M. H. Schieber, "Quantifying the independence of human finger movements: Comparisons of digits, hands, and movement frequencies," in *The Journal of Neuroscience*, 2000, pp. 8542–8550.

[50] Leap Motion. Leap motion coordinates system. Leap Motion. [Online]. Available: https://developer.leapmotion.com/documentation/cpp/devguide/Leap_Coordinate_Mapping.html?proglang=cpp

[51] Leap Motion. Leap motion hands. Leap Motion. [Online]. Available: https://developer.leapmotion.com/documentation/cpp/devguide/Leap_Hand.html

[52] X. Yi, C. Yu, M. Zhang, S. Gao, K. Sun, and Y. Shi, "Atk: Enabling ten-finger freehand typing in air based on 3d hand tracking data," in *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, 2015, pp. 539–548.

[53] T. Murase, A. Moteki, G. Suzuki, T. Nakai, N. Hara, and T. Matsuda, "Gesture keyboard with a machine learning requiring only one camera," in *Proceedings of the 3rd Augmented Human International Conference.* ACM, 2012, p. 29.

[54] H. Du, T. Oggier, F. Lustenberger, and E. Charbon, "A virtual keyboard based on true-3d optical ranging," in *Proceedings of the British Machine Vision Conference*, vol. 1, no. AQUA-CONF-2006-002, 2005, pp. 220–229.

[55] C. Divossen. Leap motion virtual keyboard. [Online]. Available: https://www.youtube.com/watch?v=9cBNvu-s0vo

[56] CorvidDude. Vr hex keyboard using leap motion. [Online]. Available: https://www.youtube.com/watch?v=ZERwYJVZOgk

[57] DexType. Dextype for the leap motion. [Online]. Available: https://www.youtube.com/watch?v=VrQRpJd8I38

[58] Wikipedia. Touch typing. [Online]. Available: https://en.wikipedia.org/wiki/Touch_typing

[59] Leap Motion. Leap motion sdk. Leap Motion. [Online]. Available: https://developer.leapmotion.com/get-started

[60] Atheer. Atheer labs. Atheer Labs. [Online]. Available: https://www.atheerlabs.com/

[61] Microsoft. Microsoft hololens. Microsoft. [Online]. Available: https://www.microsoft.com/microsoft-hololens/en-us

[62] Google. Project soli. Google. [Online]. Available: https://atap.google.com/soli/

[63] A. Colley, J. Väyrynen, and J. Häkkilä, "In-car touch screen interaction: Comparing standard, finger-specific and multi-finger interaction," in *Proceedings of the 4th International Symposium on Pervasive Displays.* ACM, 2015, pp. 131–137.

[64] T. Kuribara, Y. Mita, K. Onishi, B. Shizuki, and J. Tanaka, “Handyscope: A remote control technique using circular widget on tabletops,” in *International Conference on Human-Computer Interaction.* Springer, 2014, pp. 69–80.

[65] S. Murugappan, N. Elmqvist, K. Ramani *et al.*, “Extended multitouch: recovering touch posture and differentiating users using a depth camera,” in *Proceedings of the 25th annual ACM symposium on User interface software and technology.* ACM, 2012, pp. 487–496.

[66] A. Colley and J. Häkkilä, “Exploring finger specific touch screen interaction for mobile phone user interfaces,” in *Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures: the Future of Design.* ACM, 2014, pp. 539–548.

[67] P. Mistry and P. Maes, “Sixthsense: a wearable gestural interface,” in *ACM SIGGRAPH ASIA 2009 Sketches.* ACM, 2009, p. 11.

[68] S. Sridhar, A. M. Feit, C. Theobalt, and A. Oulasvirta, “Investigating the dexterity of multi-finger input for mid-air text entry,” in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems.* ACM, 2015, pp. 3643–3652.

[69] N. Sae-Bae, K. Ahmed, K. Isbister, and N. Memon, “Biometric-rich gestures: a novel approach to authentication on multi-touch devices,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* ACM, 2012, pp. 977–986.

List of Publications

Journals

Hani Karam and Jiro Tanaka.
An Algorithm to Detect Midair Multi-Clicks Gestures,
The Transactions of Human Interface Society Vol.19, No.3, 2017.
(Accepted, to appear)

Conference proceedings

Hani Karam and Jiro Tanaka. “Two-Handed Interactive Menu: An Application of Asymmetric Bimanual Gestures and Depth Based Selection Techniques,” *Proceedings of the 14th International Conference on Human-Computer Interaction (HCI International 2014)*, pp. 187-198, Crete, Greece, June 22-27, 2014.

Hani Karam and Jiro Tanaka. “Finger click detection using a depth camera,” *Proceedings of the 6th International Conference on Applied Human Factors and Ergonomics and the Affiliated Conferences (AHFE 2015)*, pp. 5381-5388, Las Vegas, USA, July 26-30, 2015.