平成 26 年度

筑波大学情報学群情報科学類

卒業研究論文

題目
Study on character input frameworks using one-handed gestures

主専攻　　ソフトウェアサイエンス主専攻

著者　　Mahmood Faisal

指導教員　田中 二郎 志築 文太郎 高橋 伸 三末 和男

Abstract

With the advent of head-mounted devices (HMD) like Google Glass, there is a new upcoming trend hitting the market. HMDs are devices mounted in front of the eye consisting of a non-transparent or semi-transparent display, and capable of wireless communication with other mobile devices. Uses of current generation HMDs are very limited however, mostly due to a lack of adequate input modality. Most HMDs use a small-area touch sensitive surface, voice commands or a complementary smartphone as input. This limits HMDs text input to be performed with the help of a smartphone, since voice commands in public places and even in personal quarters tend to be awkward and quirky, and the small built-in touch surface provides very little room for text input. For HMDs to grow as an independent device rather than a supplementary device for a smartphone, a text input system that does not use voice commands or an onscreen keyboard is necessary. For this research we have considered the use of one-handed gestures to work as a character input modality for use with such devices.

Gesture recognition can be performed using depth sensors that can be attached to the user's wrist or chest, allowing text input without having to wear or hold any device in the hand. In addition, by using one-handed gestures for character input, users would be able to perform text input on devices such as HMDs with little hindrance to daily activities. It would also allow users to maintain their awareness of surroundings without having to look at an input device, which is crucial during walking in public or driving. In this research, a number of hand gesture frameworks have been designed for use in such an input system. The focus of the study was to design gestures that use subtle yet distinct and recognisable finger motions. This is meant to provide users with a gesture framework that is easy to learn and efficient in terms of input speed. Although due to technical limitations in terms of accurate and consistent hand-tracking equipment, a complete system has not been developed for this study, experiments have been performed to evaluate the effectiveness of these frameworks from a design perspective.

From the results of the study, we have concluded that, although not on par with conventional input solutions, the framework designs still proved to be decently effective and easy to learn, and can become prospective candidates for use with gesture-based character input solutions for HMDs in the future.

Acknowledgements

My sincerest gratitute to my project advisor Prof. Jiro Tanaka, for providing me with numerous valuable advice and guiding me throughout this research. I am grateful to associate professors Kazuo Misue, Buntaro Shizuki and Shin Takahashi for providing me with valuable advice especially during the mid-term presentation and for reading my paper, and also to assistant professor Simona Vasilache for providing me encouragements during my research. I am also grateful to Daiki Yamaji, Yu Hayakawa and Kazuki Tada for their help in a very critical moment, and also thank Testsuya Abe, Yoshitomo Fukatsu, and everyone in the NERF - RINUX group for their advice and help on numerous accounts.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Mobile technology

1.1.1 Cell phones

Cell phones have been around for as long as since 1973, and was an evolution from analog radio communications used in ships and trains. The earliest cellphones were relatively big hand-held devices with physical buttons, and used primarily for wireless verbal communication over a cellular network without any functionality for text messages. It was two decades later that SMS was introduced to the public. SMS are short messages consisting a maximum of 160 characters. SMS was a cheaper form of communication compared to the other forms, and thereby gained popularity amongst cell phone users [1]. However, cell phones in those days used a telephone keypad layout for the keyboard. Compared to regular keyboards used with personal computers, the telephone keypad was a much slower alternative. However, due to lack of available space for a full sized keyboard, this layout seemed to have the most commonly accepted at the time and people gradually became to familiarize themselves with this layout. Even today there are many people opting to use this layout due to familiarity. And although, miniature-sized physical QWERTY keyboards were eventually implemented within cell phones, it was quickly overshadowed by the advent of smartphones.

Smartphone was the evolution from traditional cell phones. It had no physical buttons for the keypad or keyboard, but instead implemented an onscreen keyboard

usable via touch interaction. It also offered better computational capabilities allowing the cell phone to have a wide range of functionalities, which opened up a wider range uses for the cell phone. Due to the intuitive nature of browsing on a touch screen, use of cellular internet popuralized and text-based communication methods using emails and text-messaging services began to come into play. Text-based communication (also commonly known as texting) became the cheapest and most unrestricted form of communication, and eventually garnered huge popularity especially amongst young adults [2]. Cell phones eventually became the most commonly used communication terminal for the general people.

Whether for browsing the internet, writing emails or texting, it is neccessary to have an adequate character input system. The onscreen keyboard on smartphones was a huge change from the keyboard used in traditional phones. The first smartphone, the iPhone had a 3.5 inch screen, which meant the space allocated for the keyboard was very small. This caused the keyboard to be very cramped. Coupled with fact that the touch keyboard had no haptic feedback, the keyboard was not the easiest to use. However, larger screens and improved keyboard interface on more recent smartphones have alleviated this problem to a large extent. There has also been ongoing research on providing haptic feedback on touch screens [3] [4] [5] to make typing experience more intuitive. However, although onscreen keyboards might be adequate for use with smartphones, there is yet another upcoming trend that might require a different system.

1.1.2 Head-mounted devices

Head-mounted display (or HMD) are devices attached to the head, consisting of a non-transparent or semi-transparent display infront of the eye. There have been a number of HMD devices on the market till now, but it is with the upcoming release of devices such as Occulus Rift and Google Glass that it has began to garner popularity. HMDs such as the Google Glass, that are designed for mobile use, allows the users different smartphone functions without compromising awareness of surroundings, thus allowing for a better and more intuitive environment for performing multiple tasks. Although, HMDs might eventually take over smartphones, current generation mobile HMDs are not independent devices, requiring connection to smartphones for network connectivity, processing, storage or user input. With gradual improvement of mobile technology future HMDs might

eventually become self-sufficient in the first three sectors. User input, however, will require an alternative to the methods used in current generation HMDs. The Google Glass, for instance, allows user input either through a small-area touch surface or voice commands as the built-in input methods. This requires use of a smartphone for typing, since the area of the touch surface is insufficient and the voice commands are too quirky to use, especially in crowded areas. For this research, I have considered the use of hand gestures to be an alternative character input system primarily for use with HMDs. Using hand gestures, users do not have to rely on a smartphone for text input on HMDs. In addition, gestures do not require the user to look at a separate keyboard, and thus will not compromise users' awareness of surroundings.

1.1.3 Sensor technology

Sensor technology has undergone massive improvements in past couple of years. Nearly every current generation smartphone is equipped with sensors like accelerometers, magnetometers and gyroscope thanks to their miniature dimensions and low cost of production. Low cost sensors like these led to a boon in research on gesture recognition and tracking of different body parts [6] [7]. These low cost sensors however are not adequate in their current state for precise motion tracking. In comparison, recent availability of low cost infra-red (IR) cameras have given rise to several consumer-market motion tracking devices like Kinect, Leap Motion and DepthSense. However, this technology is still in its state of infancy, and current generation devices are unable to provide a consistent tracking and gesture recognition solution [8]. To provide a gesture-based text input system, this technology has to mature first and devices capable of consistent hand tracking have to emerge. Therefore, in this research I have omitted the technical aspects of this system and instead focussed on the design aspect of the framework of gestures that can be used.

1.1.4 Gesture framework design

In this study, a number of one-handed gestures have been designed to implement character input. A proposal is made of a system using a depth-sensing camera to be used as the sensor device for hand tracking and gesture recognition. By not

using any gloves or other accessories to track finger movements, users are allowed to perform daily activities with their hands without hindrance. The basic gesture design involves using the tip of the thumb to tap, flick or swipe on fingers of the same hand to make selection of desired input character. This provides the user with a haptic feedback that is more natural than using touch-based onscreen keyboards. Also the haptic feedback allows users to perform text input without having to look at their hands. It is also meant to train the muscle memory of users to get familiar with the layout of the design, and with practice to make faster inputs using the gesture system.

Chapter 2

Methodology

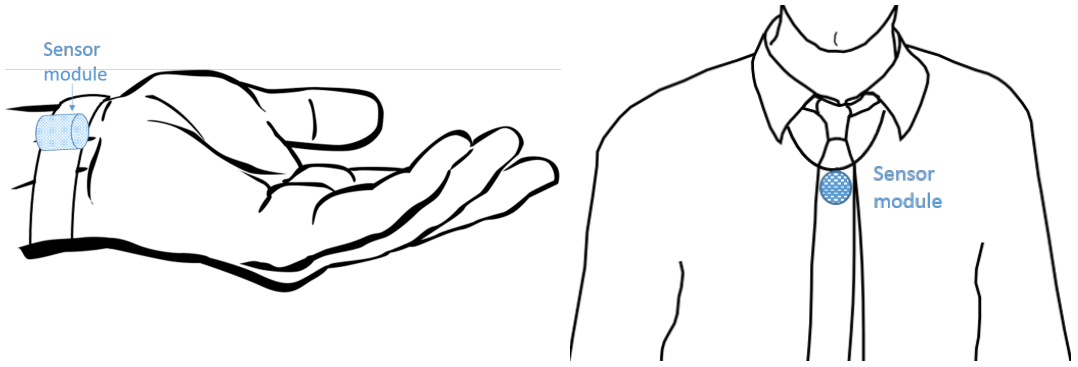
2.1 Operation

In this research, we propose the use of one-handed gestures for character input on a HMD. Users should be able to use either the right or left hand for input. A depth-sensing camera can be used to capture these gestures for processing and recognition. The gestures can be used either when the user is stationary or in motion. This allows the user to type while performing other actions with minimum hindrance. The gestures are not affected, and thus independent of the orientation or motion of the user as a whole. To prevent accidental typing and recognition of unintended gestures, a simple initiation system can be implemented using voice commands. By issuing this command, the input system will initiate and start tracking hand motion for character input.

2.1.1 Proposed setup

The proposed system involves a depth-sensing camera to capture the motion of the gesture hand. The motion images are then digitally processed to provide real-time hand-tracking and gesture recognition. We propose two different setup options for the camera. The camera positioned either under the wirst of the gesture hand (see figure 2.1a), or infront of the user's chest (see figure 2.1b).

If positioned under the wrist, it can be attached to the underside of a wristwatch or a bracelet and should be facing forward along the direction of the arm towards



(A) Sensor module positioned underneath the wrist, attached to a bracelet

(B) Sensor module positioned on the chest, hung as a necklace

FIGURE 2.1: The sensor module used for hand tracking can be positioned either on the underside of the wrist (A), or on the chest (B). If positioned underneath the wrist, users can perform text input using hand gestures regardless of hand position or orientation. If positioned on the chest, users have to raise their hand to chest level and perform hand gestures infront of the chest.

the hand. This setup will allow the user to use gestures reagardless of orientation and position of the hands. However, to prevent occulusion of fingers, the user has to bend the hand slightly inward while performing the gestures.

The second setup involves the camera to be positioned on the chest, facing forward. The camera can be setup as a necklace device, hung from the neck. This setup will require the user to lift their hands to chest level while performing the gestures. Althoug, this setup might cause an unnatural and slightly awkward feel while performing the gestures, the occulusion of fingers in this setup is relatively minimal.

Both of these setups are proposals for setups that can be used in this system. In this research, we ommitted the technical aspects of this system and focussed solely on the design aspect of the gestures in order to test the usability of one-handed gestures in such scenarios.

2.1.2 Gestures

One-handed gestures are performed to select a character for input. For this research, we used modern Roman alphabets as the alphabet system. While performing the gestures, the tip of the thumb acts as the ***pointer***. Selection of a character is made by either tapping, swiping or flicking with the tip of the thumb on the top or front side of a finger on the same hand. The character is selected according

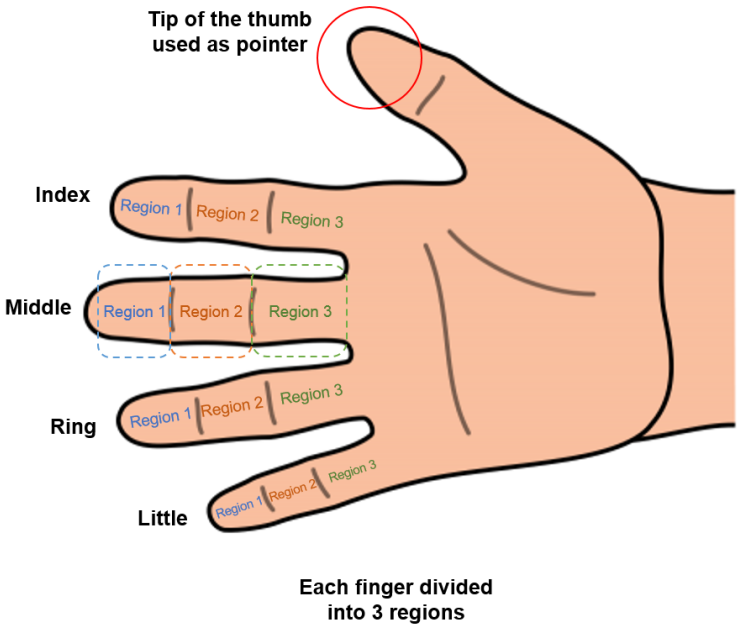


FIGURE 2.2: The figure shows the layout structure imposed on the hand. Each finger is divided into three regions, with each region consisting of a character group. The tip of the thumb is used a pointer to make selections of a character within a character group.

to a layout that is imposed on the fingers of the gesture hand, where each finger has three ***regions*** allocated to it, the regions being separated at the joints of the finger (see figure 2.2). Each region has a group of characters, hereafter reffered to as ***character groups***, allocated to it, from which the selection is made. Two different layouts have been used in this study, with different methods of selection in order to test and evaluate a suitable candidate.

The reason for choosing a one-handed gesture framework instead of a two-handed one is to allow users to type with relatively less hindrance while performing other functions such as cooking, driving, etc. This gesture system also does not require long travel distance for fingers during input, and is meant to reduce fatigue on fingers thus allowing for relatively longer sessions of typing and also improve typing speed. In addition, in this system, no gloves or other wearable accessories such as rings need to be worn on the hand, thus allowing the hands to perform regular activities while providing haptic feedback on each input to train muscle memory and allow for more precise, natural and faster typing with each iteration of use.

2.2 Frameworks

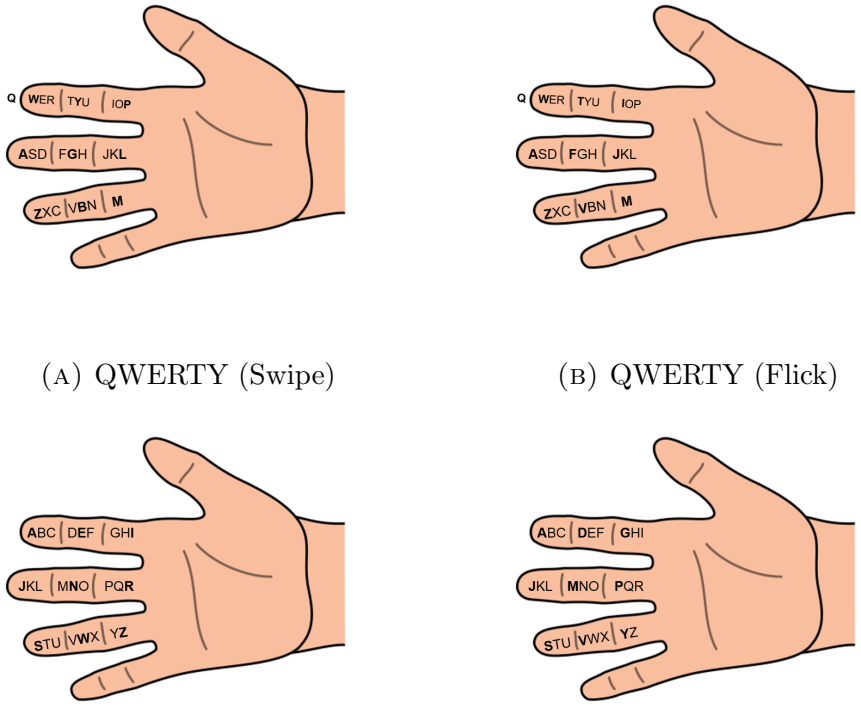
In this research, four frameworks have been designed and used for the study, consisting of two layouts (***QWERTY*** layout, and the ***Traditional keypad*** layout), with each employing one of two methods of selection (***Swipe*** method, and ***Flick*** method) (see figure 2.3).

Each layout has a different character group allocated to each region of the fingers, and one letter amongst each character group is chosen to be the ***base letter*** of that character group (represented as bold letters in the figure 2.3). To select a base letter, the user has to use the tip of the thumb to tap on the region which has the character group the base letter belongs to. A non-base letter character from a character group can be selected by placing the tip of the thumb on the region the character group belongs to, and then either swiping along the finger, or flicking inward or downward depending on the method of selection used.

For both layouts the regions on the little finger have the same character groups (see section 2.2.5). The regions in this finger allows the user to access basic punctuation marks, spaces, backspaces and return, in addition to a button for toggling between capital and small letters, and switching layout with ones containing additional punctuation marks, numbers and symbols. The gestures used to access these functions are different from the ones used to select alphabets. These gestures have been explained later in this section (see section 2.2.5). Due to this study being focussed primarily on input for Roman alphabets, layout for punctuation marks, numbers and punctuation marks have been omitted.

2.2.1 QWERTY layout

This layout is based on the standard keyboard layout used in QWERTY keyboards. The QWERTY keyboard has been around for decades, and the reason for choosing such a layout is to use the familiarity of this layout to make it easier for the general people to acclimitize to such a gesture system. This layout maps the entire QWERTY keyboard to the regions on the index, middle and ring finger, with the exception of **Q** (see figure 2.3a and 2.3b). All the other alphabets are mapped to the regions on the fingers, with each character group consisting of a maximum of three letters. For each character group to have a maximum of three



(A) QWERTY (Swipe)

(B) QWERTY (Flick)

(C) Traditional keypad (Swipe)

(D) Traditional keypad (Flick)

FIGURE 2.3: The four frameworks designed for the study. The base letters are different for each method of selection in order to suit the modality of each method.

letters without compromising the layout design, the letter **Q** has been instead mapped to the tip of the index finger. The letter **Q** is selected by tapping on the tip of the index finger with the tip of the thumb, regardless of the method of selection used with the layout.

The reasoning behind imposing the allocation of a maximum of three letters per character group has been explained later in the section (see sections 2.2.3 Swipe method and 2.2.4 Flick method).

2.2.2 Traditional keypad layout

This layout is based on the telephone keypad layout used in telephones and on older generation cellphones. This keyboard layout has also been around for decades, and a large number of people have adequate familiarity with this layout due to extensive use on past generation cellphones. The telephone keypad also has the added

benefit of congruity with the 3×4 layout structure (i.e. 4 fingers with 3 regions per finger) used in this gesture system, allowing for a more familiar transition for users of the telephone keypad layout. In this layout, the letters are distributed amongst the character groups in the index, middle and ring fingers, in alphabetical order, starting from the first region on the index finger (see figure 2.3c and 2.3d). Although, the allocation of the starting point to the second region of the index finger would seem like a more natural implementation of the telephone keypad, the change has been made to make sure each character group has a maximum of three letters allocated to it, similar to that in the QWERTY layout. The reasoning behind this has been explained later in the section (see sections 2.2.3 Swipe method and 2.2.4 Flick method).

The reasoning behind this has been expained later in the section

2.2.3 Swipe method

In the Swipe method, base letters are selected by tapping on the corresponding region with the tip of the thumb, as explained before (see section 2.2). In this method, each character group consists of a maximum of three letters, and the base letter is set to be the n^{th} letter in the character group, where n represents the region number the character group belongs to. For instance, the base letters of the second region on the index, middle and ring fingers are the second letter of the corresponding character groups, and similarly, the base letters of the first region on these fingers are the first letter of the groups (see figure 2.3a and 2.3c). The exception to this is for the third region on the ring finger, which has only one letter(for the QWERTY layout) or two letters (for the Traditional keypad layout) allocated to it. The rightmost letter in this character group has been chosen as the base letter.

To select a letter other than the base letter in a character group, the user has to place the tip of the thumb on the corresponding region and swipe along the finger. If the user wants to select the n^{th} letter letter in a character group that does not happen to be the base letter of the group, the user has to place the tip of the thumb on the corresponding region, and without removing the thumb, swipe along the finger to the n^{th} region of the finger. The exception to this is yet again the third region on the ring finger. Since there is only one letter allocated to it for the QWERTY layout, the selection of the letter is made like that of other base

letters. For the Traditional keypad layout, the rightmost letter, being the base letter, can be selected like other base letters, whereas to select the first letter, the user has to swipe to the second region of the finger. Some example scenarios have been shown in figure 2.4.

For this method to work, character groups cannot have more than three letters, since each finger consists of three regions and users would be unable to select a fourth letter in a character group.

2.2.4 Flick method

In the Flick method, similar to the Swipe method, base letters are selected by tapping on the corresponding region with the tip of the thumb, as explained before (see section 2.2). In this method, each character group consists of a maximum of three letters, and the base letter is set to be the first letter in each character group. Selection of the second letter in a character group can be done by placing the tip of the thumb on the corresponding region, and flicking inward or downward once on the same region (see figure 2.5a). Similarly, to select the third letter of a character group, the user has to flick twice (see figure 2.5b).

2.2.5 Character groups on the little finger

The three regions on the little finger consist of unique character groups called ***action buttons*** (or ***AB***) (see figure 3.3). Regardless of the method of selection used for the layout, characters or functions allocated to an action button can be accessed using the tip of the thumb and performing either a single tap, a double tap, a flick in the direction of the tip of the finger, or a flick in the direction of the palm of the hand.

As a sidenote, cycling through layouts changes the layout of the index, middle and ring fingers to access input of numbers, additional punctuation marks and symbols. The character groups on the little finger, however, remain the same.

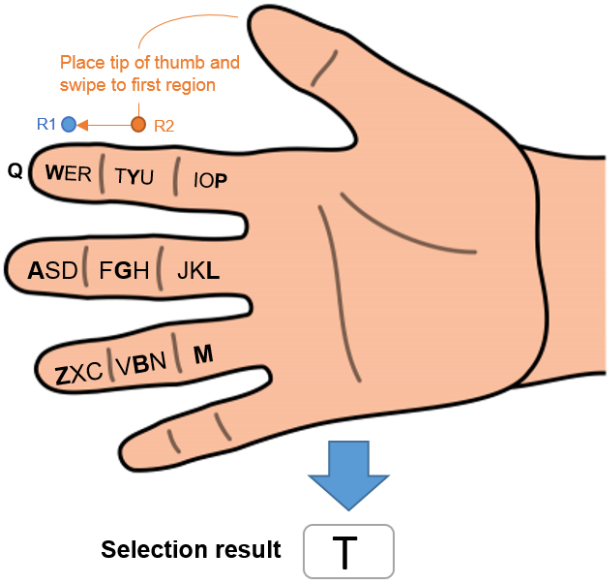
Button	**Single tap**	**Double tap**	**Flick towards tip**	**Flick towards palm**
AB1	Space	Return	Backspace	Tab
AB2	Comma (,)	Period (.)	Exclamation mark (!)	Question mark (?)
AB3	Toggle Caps on/off	Cycle layouts	None	None

TABLE 2.1: The table shows the functions and characters in each action button on the little finger and their selection methods

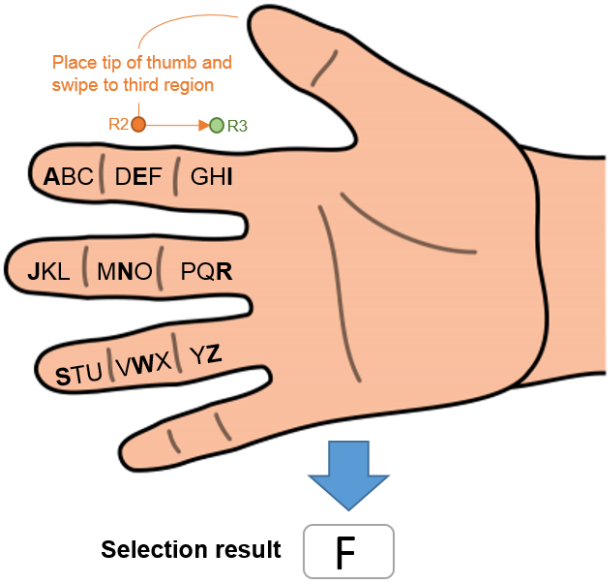
2.3 Comparison to past works

HMD is still a relatively new technology, and there has not yet been many research on input systems for the HMD especially using hand-gestures. Regarding research works on gesture framework design for input on cellphones however, there have been several works the field. In a research, for example, tilt sensing was introduced to a telephone keypad to select letters using a single tap on a key, the selection made being based on the tilt angle of the phone [9]. The tilt albeit an interesting idea, might not be very practical for hand-based gesture frameworks, since the continuous rotations of the arm might cause fatigue and also cause attention in public places.

Another mentionable research was done on a system using a glove to map the telephone keypad directly on the fingers called The Finger-Joint-Gesture palm-keypad [10]. Selections were performed by tapping with the tip of the thumb on the inner faces of the fingers. Quite similar to the Traditional keypad (Flick) framework used in this study, the reasoning behind using flick instead of taps was to use flicks only for selection, and touch or tap with the tip of the thumb to be used for visual feedback for verification of character groups present on the regions of the fingers. In addition, flicks are more pronounced and relatively more easily recognizable by hand-tracking devices than taps. The Traditional keypad layout used in this study, instead of being a direct transition of the telephone keypad, has been altered to garner to the need of the methods of selection used.

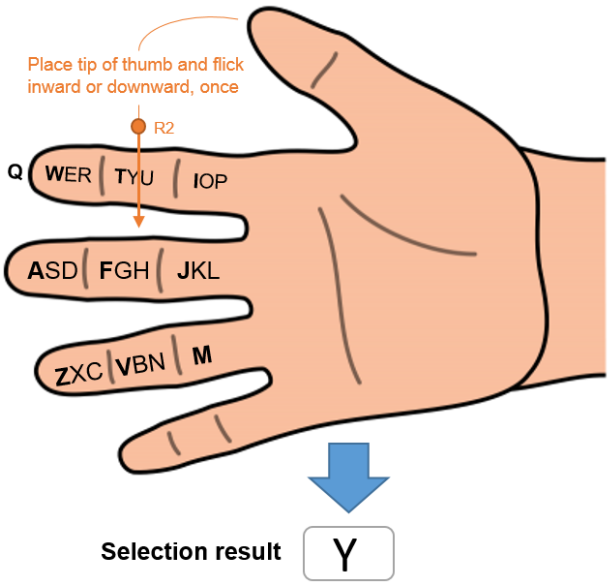


(A) For the QWERTY layout, to select the first letter of the character group in the second region on the index finger, T, the user has to place the tip of the thumb on the region and swipe to the first region on the finger.

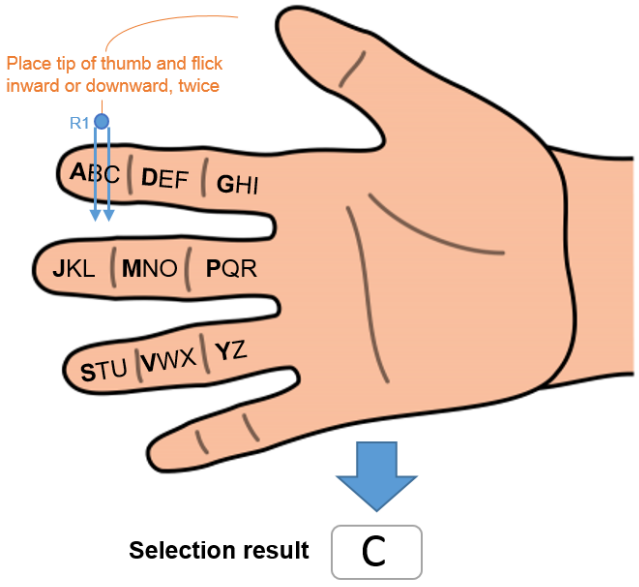


(B) For the Traditional keypad layout, to select the third letter of the character group in the second region on the index finger, F, the user has to place the tip of the thumb on the region and swipe to the third region on the finger.

FIGURE 2.4: The figure shows some example scenarios while using the Swipe method



(A) For the QWERTY layout, to select the second letter of the character group in the second region on the index finger, Y, the user has to place the tip of the thumb on the region and flick inward or downward, once.



(B) For the Traditional keypad layout, to select the third letter of the character group in the first region on the index finger, C, the user has to place the tip of the thumb on the region and flick inward or downward, twice.

FIGURE 2.5: The figure shows some example scenarios while using the Flick method

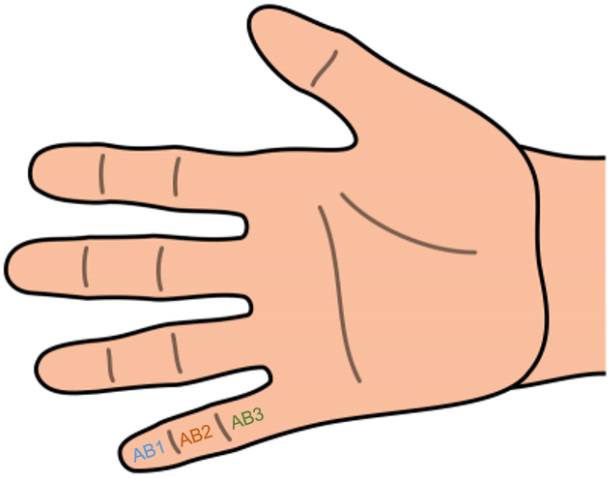


FIGURE 2.6: The figure shows the layout imposed on the little finger. AB refers to action buttons, and are special character groups with unique method of selection.

Chapter 3

Study experiments

A study was performed to evaluate the difficulty curve involved in learning and the speed of character input that can be performed using the different frameworks designed. A complete gesture-based character input system using Leap Motion as the sensor device was meant to be developed for this study. However, hand-tracking with the Leap Motion device involving acute finger movements proved to be inconsistent at times, and thereby prompted us to abandon development of the complete input system and instead opt for a more forgiving ***visual feedback application*** (see section 3.1.2). The visual feedback application is a rudimentary application developed that uses the Leap Motion to track the thumb's position and display different character groups on a finger when the thumb hovers near that particular finger. This proved to be adequate for the study, since the aim of this study was to evaluate the effectiveness of the design of the frameworks instead of the technical aspects of the input system.

The study consisted of two types of experiments, and was performed on a group of three male and two female (a total of 5) participants. After the experiments, the participants were asked about their impressions on each framework used in the experiments. All of the participants were in the age group 20-35 years old, with adequate grip on english language vocabulary. The particants were regular smart-phone users and had ample familiarity with the onscreen QWERTY keyboard as well as former users of the telephone keypad in older cellphones. The participants had no prior experience with gesture-based text input systems.

3.1 Instruments and applications used

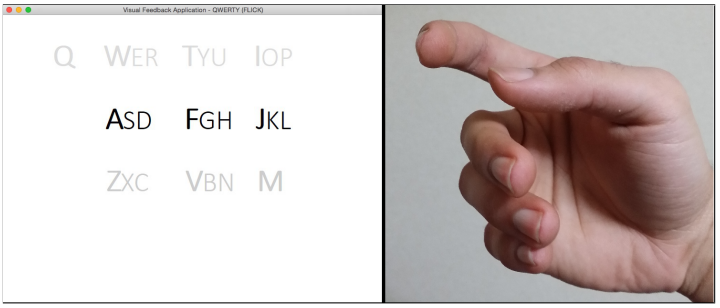
3.1.1 Instruments

For this study, a Leap Motion sensor has been used to track hand motion. A Mac mini computer with 3rd generation i7 processor, built-in Intel graphics and ssd harddrive and a 27 inch screen has been used for processing and visual onscreen feedback. A Galaxy Note 3 was used as a camera to record videos of the participants performing the gestures. A digital stopwatch has been used to count time during the experiments.

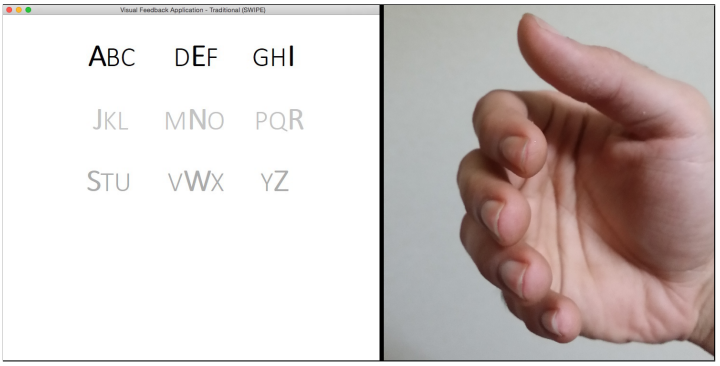
3.1.2 Visual feedback application

A rudimentary application has been developed using Python to provide a visual feedback for the participants during the experiments. This feedback is triggered when a hand is placed infront of the Leap Motion camera, and the tip of the thumb hovers near a finger. The application UI shows an image of the mapping of the characters for the framework, with the characters on the finger closest to the tip of the thumb being highlighted (fig. 3.1). This allowed the participants to associate and familiarize with the frameworks, and provided visual cues to the participants during the experiments.

Although the sensor used in Leap Motion is able to detect slight changes in position, in our experience the hand-tracking implemented in the Leap Motion's curent software development kit (as per version. SDK 2.1.3) proved to be quite inconsistent.Despite the sensor being capable of detecting slight change in hand position, change in hand posture, however, sometimes gave abrupt results, which could cause a number of rogue inputs and thus hamper the results of the experiments. This led to our decision of omitting a fully functional gesture-based character input system using the Leap Motion, and instead opt for a more forgiving visual feedback system. In this system, the application infers the finger closest to the thumb, and is triggered when the distance between a finger and the thumb is less than a threshold value that has been designated beforehand. This threshold value has been determined after several trial and errors until a suitable value has been obtained.



(A) Visual feedback application UI for the QWERTY (Flick) framework (left), obtained when the tip of the thumb is hovering near the middle finger (right).



(B) Visual feedback application UI for the Traditional keypad (Swipe) framework (left), obtained when the tip of the thumb is hovering near the index finger (right).

FIGURE 3.1: The above figure shows the visual feedback application UIs for two different frameworks. In the UI, the letters are grouped together to represent association to the same character groups, and the layout showed is a direct image of the layout mapped on the fingers for that particular framework. In addition, the letters with larger, bold fonts represent the base letters of the corresponding regions.

While using this feedback application, although hand-tracking inconsistencies still led to incorrect feedbacks at times, the effect on the results were not as detrimental as would have been if a complete input system was used instead.

3.1.3 Setup

The Leap Motion was attached to the user's chest facing forward, and was connected to the computer via usb. The participants were asked to sit in a chair infront of the computer screen and perform the tasks for the experiments. For each experiment a list of random sentences were shown to the participant, displayed on the left side of the screen, and the visual feedback application displayed on the right. The participants were given clear instructions on the task to be

performed before each experiment, and provided with one to two minute breaks inbetween each experiment.

During the experiments, videos of the participants performing the gestures were recorded from a close distance with a clear view of the gesture hand and fingers. The videos were captured in high-definition 720p at 60 frames per second, and were later analysed to determine the number of correct and wrong gestures performed during the experiments.

3.2 Procedure

Two experiments were performed for each layout: Familiarity, and speed. For each layout, the familiarity experiment was performed first and then the speed experiment for the same layout, before moving on to the next layout and performing these experiments in the same order.

3.2.1 Familiarity experiment

The purpose of the familiarity experiment was to determine the difficulty of the learning curve involved for each layout. It also serves as a way for the participants to familiarize themselves with the layouts in order to perform the speed experiment. In this experiment, an adequately long list of randomly generated sentences were displayed on the screen. The participants were then asked to tap on the region on their fingers that corresponded to each letter in the sentences. Instead of performing swipe or flicks to make character selections, the participants were instructed only to point the region where a particular letter belonged to. The reason for this was to strengthen the muscle memory of the participants to increase their familiarity with the layout.

This experiment was performed for 5 minutes, and iterated over 3 trials for each participant, with small breaks inbetween iterations. The results obtained from the videos were later analysed to determine the number of successful ***hits*** (i.e. regions pointed correctly) by each participant in the given time frame for each trial.

3.2.2 Speed experiment

The purpose of the speed experiment was to determine the speed and reliability with which a participant can use a framework for text input. This experiment is performed twice per layout, one for each method of selection. Before each of these experiments, participants were provided with a list of 10 randomly generated sentences, and instructed to take their time and type those sentences correctly using the corresponding framework. The reason for this was to familiarize participants with each method of selection before the experiment. In the experiment, an adequately long list of randomly generated sentences were again displayed on the screen. Participants were then asked to use the corresponding framework, to input the letters in the sentences shown. They were asked to complete as many sentences as possible in a 2 minute timeframe. They were instructed not to correct any errors (i.e. missed inputs) but instead to continue typing the next letter. For the sake of emphasising on the alphabets, punctuation marks are omitted from the sentences, although space and newline functions are taken into account. In addition, letter case was also not taken into consideration, and participants were asked to perform similar gestures for both upper case and lower case letters.

Each experiment is iterated 3 times, with small breaks inbetween each iteration. The results obtained from the videos were later analysed to determine the number of inputs attempted and errors made for the participants. The number of correct inputs was then deduced with the following equation:

$$\text{Number of correct inputs} = \text{Number of attempted inputs} - (\text{Number of errors} \times 3)$$

The reasoning behind this is that every error costs three inputs to be made; one being the missed input, another being a backspace, and the last being a correct input.

3.3 Evaluation

During the experiments, high-definition videos of the participants' hands were captured while performing the given tasks. The videos were later analysed to evaluate the number of correct and missed inputs performed by the participants during the experiments. The lack of an automated input system prompted us to

do this evaluation in person by watching the videos and checking for the number of correct and missed inputs performed by the participants. To make this process easier, we first converted the random sentences provided to the participants into ***gesture codes***. In the gesture code, each character is represented using three numbers (for example, 332), and these triplets are separated using hyphens.

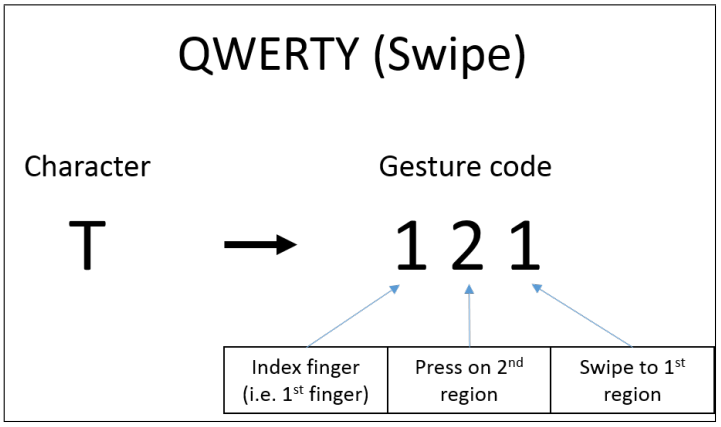
The gesture code for a character defines the gesture to be performed to input that character, and, as a result, varies with the framework being used for input. The first number of the triplet signifies the finger on which the gesture has to be performed; "1" used to indicate the index finger, "2" for the middle finger, "3" for the ring finger and "4" for the little finger. The second number indicates the region of the finger on which the character exists; "1" indicating region 1, "2" for region 2, and "3" for region 3 (see fig. 2.2 for illustration on "regions"). The third number of the triplet has two different definitions depending on the method of selection used.

For the swipe method, the third number indicates the region number on which the swipe has to end on. For example, in the QWERTY (Swipe) framework, to select the character "T", the user has to press on the region 2 of the index finger, then swipe to region 1 and release (see fig. 2.4a). In this case, the third number of the code would be "1", and the gesture code for the character would be "121" (fig. 3.2a).

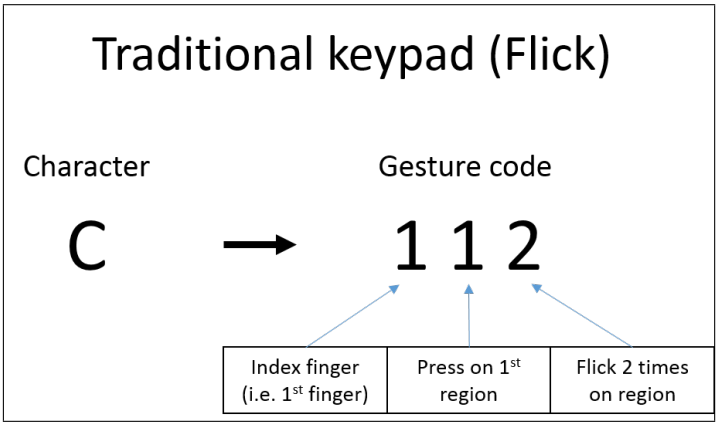
For the flick method, the third number indicates the number of flicks the user needs to perform to select the character. For example, in the Traditional keypad (Flick) framework, to select the character "C", the user has to flick 2 times on region 1 of the index finger (see fig. 2.5b). In this case the third number of the code would be "2", and the gesture code for the character would be "112" (fig. 3.2b). For characters that only reqire a tap to select, the third number would be "0".

The only exception to this coding rule is the character "Q" on the QWERTY layout which is defined by the code "100". In addition the code for the function"space" is "400", and for the function "newline", the code "401" has been used.

The conversion of the sentences used in the experiments into gesture codes was done using an application, ***gesture code generator***, that has been developed to generate codes from alphabets in a text file. The generator outputs the gesture code and saves it in another text file. It was developed using Python and require



(A) Gesture code example for QWERTY (Swipe) framework



(B) Gesture code example for Traditional keypad (Flick) framework

FIGURE 3.2: The above figure illustrates the format of the gesture code using two examples.

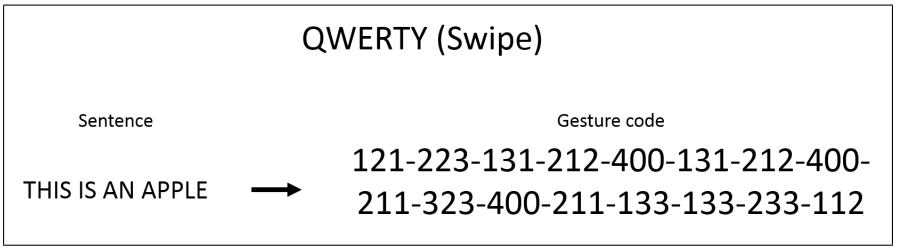


FIGURE 3.3: The figure shows an example of gesture codes generated for a given sentence for QWERTY (Swipe) framework.

Python to be installed on the computer to be able to use it. In addition, alphabets in the input text file have to be in roman capital letters for the application to work.

After the experiments, the gestures performed by the participants were analysed by watching the videos and simulatneously comparing the gestures with the corresponding code to evaluate if the gestures were correct or not. This method of

evaluation turned out to be quite time-consuming and stressful, however, lack of alternatives in the given timeframe prompted us to use this method.

Chapter 4

Results and discussion

4.1 Results

4.1.1 Familiarity experiment results

After analysis of the videos obtained in the familiarity experiment, the mean number of successful hits performed by the participants for each trial has been calculated and plotted on a graph (fig. 4.1). The results show significant increase in number of successful hits with each successive trial. For the QWERTY layout, the number of successful hits went up by 51% over the the three trials, and that of the Traditional keypad layout went up by 42%.

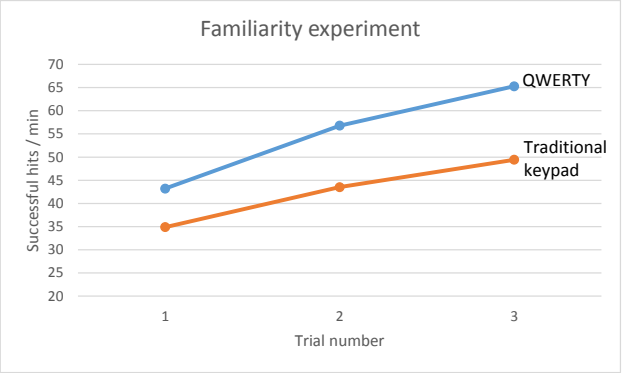


FIGURE 4.1: The above figure shows the results obtained from the familiarity experiment. The x-axis represents the trial number, while the y-axis represents the average successful hits for the participants.

4.1.2 Speed experiment results

Similarly, for the speed experiment, after analysis of the videos obtained during the experiment, the mean number of correct inputs performed per minute by the participants has been calculated and plotted on a graph (fig. 4.2). The results again show increase in speed for the QWERTY layout compared to the Traditional keypad layout. On the other hand, the Flick method is found to be faster compared to the Swipe method.

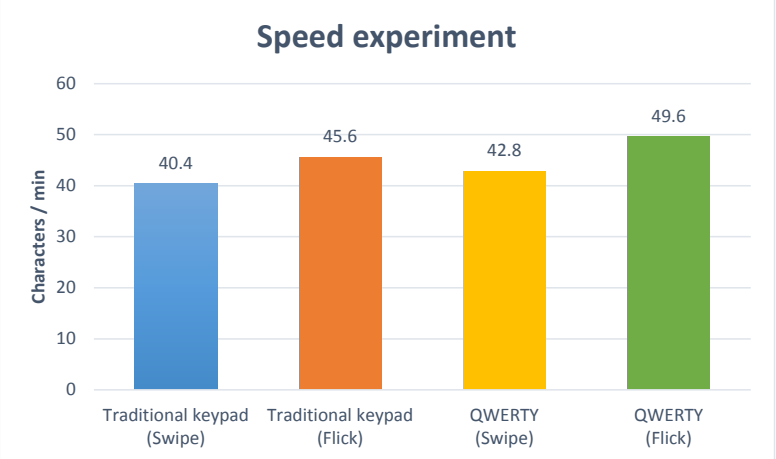


FIGURE 4.2: The above figure shows the results obtained from the speed experiment. The graph shows the average input speed of all participants for each framework.

4.1.3 Participant impressions

4.1.3.1 Layouts

Three of the five participants stated the QWERTY layout to be more natural and easily recognizable of the two layouts. A general consensus was that, in the Traditional keypad layout, the leftward shift in alphabet positions from the telephone keypad was slightly unnerving to the participants.

4.1.3.2 Methods of selection

All the participants stated the flick method to be the most easy to learn of the two methods. The swipe method was stated to be a bit complicated and difficult to master, however 4 of the participants said that the swipe method caused less stress on the fingers compared to the flick method and might be suitable for longer period of use.

4.2 Discussion

From the results obtained in the experiments (see fig 4.1 and 4.2) and individual impressions of the participants, the QWERTY layout proved to be the more natural and easily recognizable of the two layouts. This might be a result of frequent use of QWERTY onscreen keyboards on smartphones and computer keyboards by the general people. From the the familiarity experiment results, both the layouts showed significant increase in number of successful hits with each successive trial showing that people can get familiar with the layouts in very short time. According to the results and participant impressions, the QWERTY layout in particular seemed to be more easiliy recognizable, and although due to the small scope of the study only three trials were attempted for each participant, the results showed promising results with regards to easy adaptation to a gesture system using this layout. Regarding the speed experiment, although the increase in speed for the QWERTY layout over the Traditional keypad layout did not seem that sginificant, this can be attributed to the lack of familiarity with the individual methods of selection giving rise to the inconsistency.

As for the method of selection, participants seem to find the flick method to be faster and easier to understand than the swipe method. Although the input speed of flick method appears to be faster than the swipe method, participants stated the swipe method to less stressful. This a valuable point since long stretches of input might fatigue the user, especially if the user is involved in other activities that are already stressful to the hand or arms (manual labour for example). In addition, participants stated it was rather the learning curve and not the act of performing the gesture itself that was difficult. This means that although the method is difficult to get used, it might be a faster alternative in the long run since theoretically motions required per input are have less travel and thereby cause less stress.

Figure 4.3 shows the results obatined in the speed experiment in terms of word count instead of character count compared to currently used text input solutions. This provides a better perspective of how a gesture system using these frameworks might fare against a full-sized physical or smartphone onscreen keyboard. Compared to a smartphone onscreen keyboard (with an average typing speed of 21.8 words per minute [11]), even the QWERTY (Flick) framework with the maximum speed obtained in the experiments, is less than half of that speed. However, when we consider the average typing speed on full-sized physical QWERTY keyboards (around 64 words per minute [12]), even the smartphone onscreen keyboard provides a fraction of the speed. However, this gesture system, just like the onscreen keyboard on smarthphones is not designed for long stretches of typing but instead for the text input used in short mails, text messages, browsing and so on. For such a purpose this speed might be considered acceptable especially when used with HMDs which currently do not have any proper text input modility other than a smartphone.

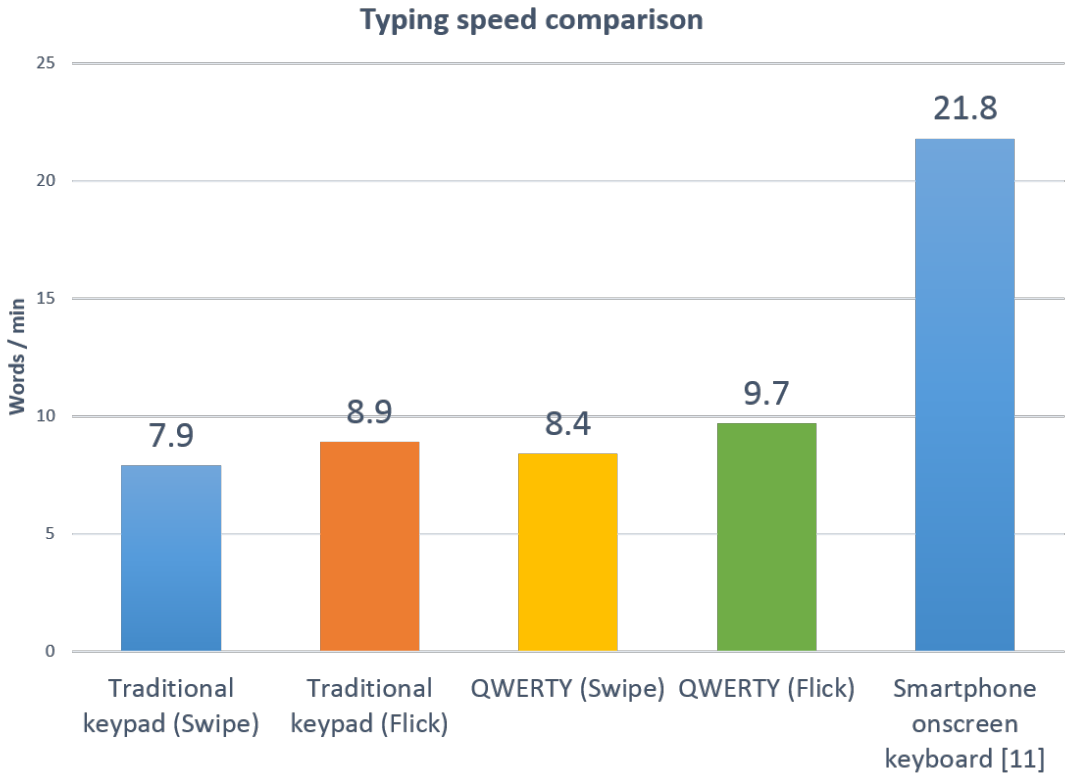


FIGURE 4.3: The above figure is a derived representation of the results obtained in the speed experiment compared to smartphone onscreen keyboards. In this figure, character count from the speed experiment results has been converted to word count with 5.1 characters per word as can be considered an average in the english language [13]

Chapter 5

Conclusion

In this research, we proposed the use of a one-handed gesture system for character input on HMD. We then designed four gesture frameworks for use with such a system. Using layouts already being used on conventional text input systems such as the physical QWERTY keyboard and traditional mobile phones, we tried to use the familiarity of the layouts in a one-handed gesture based input scenario. The results show that the familiarity of the layouts indeed led users to perform input at decent speed. The QWERTY layout in particular stood out as being more familiar due to frequent daily use both on smartphones and physical keyboards.

In addition, we also evaluated the effectiveness of two different methods of selection designed for the layouts. Among these, the Flick method seemed to be easier to learn and use, although participants of the experiments did report Swipe method to be less stressful on the hand. This leads to our conclusion that the Swipe method might be a better choice for prolonged use provided users can take their time in familiarizing with the method. The Flick method, on the other hand, is the easier method to learn and readily usable even for beginner users.

Due to the short time involved in the research, and a lack of device capable of consistent hand-tracking, a full-scale implementation and study of such a system could not be not accomplished. Given the promising results from the design evaluation, if such a system could be implemented with consistent and precise hand-tracking, we believe it can be a suitable character input solution for use with HMDs, and by doing that, enable HMDs to be one step closer to becoming an independent mobile device. Therefore, as for future works on this subject, we wish to futher the study and implement a suitable hand-tracking solution for the system. In

addition, we need to further investigate into improvements on these frameworks, or testing completely new ones on a full-scale system with real-time hand-tracking and proper visual feedback on a HMD.

Bibliography

[1] James E. Katz. Handbook of mobile communication studies. page 22, 2008. URL http://en.wikipedia.org/wiki/Mobile_phone.

[2] Shintaro Okazaki. What do we know about mobile internet adopters? a cluster analysis. *Information & Management*, 43:127141, March 2006. URL http://www.sciencedirect.com/science/article/pii/S0378720605000340.

[3] In Ian Oakley and Stephen Brewster, editors, *Haptic and Audio Interaction Design*, volume 4813 of *Lecture Notes in Computer Science*. 2007. ISBN 978-3-540-76701-5.

[4] Stephen Brewster, Faraz Chohan, and Lorna Brown. Tactile feedback for mobile interactions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, 2007.

[5] Yvonne Jansen, Thorsten Karrer, and Jan Borchers. Mudpad: Tactile feedback and haptic texture overlay for touch surfaces. In *ACM International Conference on Interactive Tabletops and Surfaces*, ITS '10, 2010.

[6] Juha Kela et al. Accelerometer-based gesture control for a design environment. *Personal Ubiquitous Comput.*, 10(5), July 2006.

[7] uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 5(6):657 – 675, 2009. PerCom 2009.

[8] Frank Weichert, Daniel Bachmann, Bartholomus Rudak, and Denis Fisseler. Analysis of the accuracy and robustness of the leap motion controller. *Sensors*, 13(5):6380–6393, 2013. URL http://www.mdpi.com/1424-8220/13/5/6380.

[9] Daniel Wigdor and Ravin Balakrishnan. Tilttext: Using tilt for text input to mobile phones. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*, UIST '03, 2003.

[10] *Personal Technologies*, 4(2-3), 2000. ISSN 0949-2054.

[11] Patti Bao, Jeffrey Pierce, Stephen Whittaker, and Shumin Zhai. Smart phone use by non-mobile business users. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '11, 2011.

[12] I. Scott MacKenzie Edgar Matias and William Buxton. Half-qwerty: a one-handed keyboard facilitating skill transfer from qwerty. In *In Conference proceedings on Human factors in computing systems*, CHI '93, 1993.

[13] Wolfram alpha. URL http://www.wolframalpha.com/input/?i=average+english+word+length.

Appendix A

Source code: Visual feedback application

The following shows the code used for the visual feedback application for the QWERTY (Flick) framework. Visual feedback applications developed for the other frameworks were similar to the one showed below, with the only difference being the images files used in lines 21 to 25, and the title of the UI in line 72.

```
1  import sys, math
2  sys.path.insert(0,"../lib") #Directory for the Leap Motion library
       files
3  from PIL import Image, ImageTk
4  from Tkinter import Frame, Canvas, YES, BOTH
5  import Leap
6
7  class SampleListener(Leap.Listener):
8      finger_closest = 0
9      init = 1
10     prev_im = Image.open('3.jpg')
11     im = prev_im
12
13     def on_init(self, controller):
14         print "Initialized"
15
16     def on_connect(self, controller):
17         print "Connected"
18
```

```
19      def draw(self, position):
20          if  position == 1:
21              self.im = Image.open('vfa_qf_01.jpg')
22          elif position == 2:
23              self.im = Image.open('vfa_qf_02.jpg')
24          elif position == 3:
25              self.im = Image.open('vfa_qf_03.jpg')
26
27  # "vfa_qf_01.jpg", "vfa_qf_02.jpg" and "vfa_qf_03.jpg" are images
      highlighting the character groups on first, second and third
      finger respectively, for the QWERTY (Flick) framework. Similarly,
      image files like these were prepared beforehand for use with the
      other frameworks
28
29          if self.prev_im != self.im:
30              self.refresh()
31              self.paintCanvas.image = ImageTk.PhotoImage(self.im)
32              self.paintCanvas.create_image(0, 0, image=self.
      paintCanvas.image, anchor='nw')
33
34              self.prev_im = self.im
35
36      def set_canvas(self, canvas):
37          self.paintCanvas = canvas
38
39      def refresh(self):
40          self.paintCanvas.delete("all")
41
42      def on_frame(self, controller):
43          frame = controller.frame()
44
45          for hand in frame.hands:
46              pointer = hand.fingers[0].bone(3).next_joint
47              op_minDist = 55
48
49   # "op_minDist": Distance between thumb and finger must be closer
      than this distance for feedback to trigger
50
51              hand_id = hand.id
52              confidence = hand.confidence
53              for a in range((hand_id * 10) + 1 , (hand_id * 10) + 5):
54                  finger = hand.finger(a)
55                  for b in range(1, 4):
56                      bone = finger.bone(b)
```

```
57                      if confidence > 0.5:
58                          if pointer.distance_to((bone.next_joint+bone.
    prev_joint)/2) <= op_minDist:
59                              op_minDist = pointer.distance_to((bone.
    next_joint+bone.prev_joint)/2)
60                              self.finger_closest = finger.type()
61              self.draw(self.finger_closest)
62
63
64  class VFA(Frame):
65
66      def __init__( self ):
67          Frame.__init__( self )
68          self.leap = Leap.Controller()
69          self.vfa_listener = SampleListener()
70          self.leap.add_listener(self.vfa_listener)
71          self.pack( expand = YES, fill = BOTH )
72          self.master.title( "Visual Feedback Application - QWERTY (
    FLICK)" ) # VFA for QWERTY (FLICK)
73          self.master.geometry( "800x500" )
74
75          self.paintCanvas = Canvas( self, width = "800", height = "500
    " )
76          self.paintCanvas.pack()
77          self.vfa_listener.set_canvas(self.paintCanvas)
78
79  def main():
80      VFA().mainloop()
81
82  if __name__ == "__main__":
83      main()
```

Appendix B

Source code: Gesture code generator

The following shows the code used for the application for generating gesture codes out of text files containing random sentences prepared for the study experiments (see section 3.3). The file "task.txt" (line 82) corresponds to the input file, and the file "gc.txt" (line 80) corresponds to the output file containing the codes.

```
1  Matrix = [[[0 for x in range(3)] for x in range(3)] for x in range(3)
       ]
2  try:
3      mode=int(raw_input('Enter -\n 1 for QWERTY (Swipe)\n 2 for QWERTY
       (Flick)\n 3 for Trad (Swipe)\n 4 for Trad (Flick):\n'))
4  except ValueError:
5      print "Not a number"
6
7  if mode == 1:
8      Matrix [2][2][2] = "M"
9
10 elif mode == 2:
11     Matrix [2][2][0] = "M"
12
13 elif mode == 3:
14     Matrix [2][2][1] = "Y"
15     Matrix [2][2][2] = "Z"
16
17 elif mode == 4:
```

```
18      Matrix [2][2][0] = "Y"
19      Matrix [2][2][1] = "Z"
20
21
22  if mode == 1 or mode == 2:
23      Matrix [0][0][0] = "W"
24      Matrix [0][0][1] = "E"
25      Matrix [0][0][2] = "R"
26
27      Matrix [0][1][0] = "T"
28      Matrix [0][1][1] = "Y"
29      Matrix [0][1][2] = "U"
30
31      Matrix [0][2][0] = "I"
32      Matrix [0][2][1] = "O"
33      Matrix [0][2][2] = "P"
34
35      Matrix [1][0][0] = "A"
36      Matrix [1][0][1] = "S"
37      Matrix [1][0][2] = "D"
38
39      Matrix [1][1][0] = "F"
40      Matrix [1][1][1] = "G"
41      Matrix [1][1][2] = "H"
42
43      Matrix [1][2][0] = "J"
44      Matrix [1][2][1] = "K"
45      Matrix [1][2][2] = "L"
46
47      Matrix [2][0][0] = "Z"
48      Matrix [2][0][1] = "X"
49      Matrix [2][0][2] = "C"
50
51      Matrix [2][1][0] = "V"
52      Matrix [2][1][1] = "B"
53      Matrix [2][1][2] = "N"
54
55  elif mode == 3 or mode == 4:
56      Matrix [0][0][0] = "A"
57      Matrix [0][0][1] = "B"
58      Matrix [0][0][2] = "C"
59
60      Matrix [0][1][0] = "D"
61      Matrix [0][1][1] = "E"
```

```
62      Matrix [0][1][2] = "F"
63
64      Matrix [0][2][0] = "G"
65      Matrix [0][2][1] = "H"
66      Matrix [0][2][2] = "I"
67
68      Matrix [1][0][0] = "J"
69      Matrix [1][0][1] = "K"
70      Matrix [1][0][2] = "L"
71
72      Matrix [1][1][0] = "M"
73      Matrix [1][1][1] = "N"
74      Matrix [1][1][2] = "O"
75
76      Matrix [1][2][0] = "P"
77      Matrix [1][2][1] = "Q"
78      Matrix [1][2][2] = "R"
79
80      Matrix [2][0][0] = "S"
81      Matrix [2][0][1] = "T"
82      Matrix [2][0][2] = "U"
83
84      Matrix [2][1][0] = "V"
85      Matrix [2][1][1] = "W"
86      Matrix [2][1][2] = "X"
87
88  if Matrix [0][0][0]!= 0:
89      file = open("gc.txt", "w+") # Output text file with generated
        gesture code
90      file.truncate()
91      with open("task.txt") as f: # Input text file (text file
        containing random sentences provided to the participants)
92          while True:
93              c = f.read(1)
94              if not c:
95                  print "Done"
96                  break
97
98              for n1 in range (3):
99                  for n2 in range (3):
100                     for n3 in range(3):
101                         if c == Matrix[n1][n2][n3]:
102                             if mode == 1 or mode == 3:
```

```
103                                     file.write("%d%d%d-"%(n1+1,n2+1,n3+1)
    )
104                                 else:
105                                     file.write("%d%d%d-"%(n1+1,n2+1,n3))
106
107
108             if c == " ":
109                 file.write("400-")
110             elif c == "\n":
111                 file.write("401-")
112             elif mode == 1 or mode == 2:
113                 if c == "Q":
114                     file.write("100-")
115
116     file.close()
```

Appendix C

Familiarity experiment data

Familiarity experiment

Traditional layout							
Mean per candidate	**Mean (HPM)**	Trial no.	Candidate				
			C1	C2	C3	C4	C5
174.4	**34.88**	T1	186	171	158	167	190
217.6	**43.52**	T2	226	224	198	210	230
247.2	**49.44**	T3	253	261	223	241	258

QWERTY layout							
Mean per candidate	**Mean (HPM)**	Trial no.	Candidate				
			C1	C2	C3	C4	C5
216	**43.2**	T1	231	223	197	208	221
284	**56.8**	T2	291	294	268	279	288
326.4	**65.28**	T3	329	344	299	325	335

*HPM: successful hits per minute

Appendix D

Speed experiment data

Speed experiment						
Trad Swipe						
Mean (CPM)	Total characters in 3 iterations (6 min)					
40.4	Candidate	C1	C2	C3	C4	C5
	Total	290	279	245	258	275
	Errors	14	6	8	7	10
	Correct	248	261	221	237	245

Trad Flick						
Mean (CPM)	Total characters in 3 iterations (6 min)					
45.6	Candidate	C1	C2	C3	C4	C5
	Total	314	306	277	276	309
	Errors	12	5	7	6	8
	Correct	278	291	256	258	285

*CPM: characters per minute
Mean = Total correct characters / (5*6)
i.e. 5 participants, 6 minutes

FIGURE D.1: Speed experiment : Traditional keypad layout

Speed experiment

QWERTY Swipe

Mean (CPM)	Total characters in 3 iterations (6 min)					
42.8	Candidate	C1	C2	C3	C4	C5
	Total	288	300	258	267	291
	Errors	11	8	5	6	10
	Correct	255	276	243	249	261

QWERTY Flick

Mean (CPM)	Total characters in 3 iterations (6 min)					
49.6	Candidate	C1	C2	C3	C4	C5
	Total	329	326	300	306	323
	Errors	10	5	5	6	6
	Correct	299	311	285	288	305

*CPM: characters per minute
Mean = Total correct characters / (5*6)
i.e. 5 participants, 6 minutes

FIGURE D.2: Speed experiment : QWERTY layout