TRINITY COLLEGE DUBLIN

MCS DISSERTATION

Augmented Reality Keyboard for an Interactive Gesture Recognition System

Author:
SAM GREEN

Supervisor:
Dr. Gerard Lacey

A dissertation submitted in fulfillment of the requirements for the degree: ***Masters in Computer Science*** *at the*
School Of Computer Science & Statistics

Submitted to the University of Dublin, Trinity College, May 2016

Declaration

I, Sam Green, declare that the following dissertation ("Augmented Reality Keyboard for an Interactive Gesture Recognition System") except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

Signed:

Date:

Summary

The goal of this dissertation was to develop an augemented reality keyboard which provides touchless, gesture-based typing. The system's design was motivated by recent developments in 3D camera technology which have allowed time-of-light depth sensing hardware devices to become highly accurate and generally inexpensive. Recognizing gestures in mid-air for text-entry was an important facet of this system's design, due to the unique applicability of touchless typing in a number of use cases and environments. The application was implemented for SureWash, an existing healthcare cart-based system which automates the process of hand hygiene training for hospital staff.

The system was designed to model the traditional QWERTY keyboard in terms of its graphical user interface. An augmented reality approach was also specified in the design, which would allow the interface to blend real-time video streams with virtual UI elements. The input model for the system is based around accepting swipe gestures in which words are entered per-swipe, rather than per-key pressed. It was developed using computer vision and fuzzy string matching algorithms and techniques.

The system processes depth data from the camera hardware device to segment a user's hand from the static background and detect fingertips in the scene. The depth-masked hand is used to seed an adaptive skin colour histogram for hand segmentation in the higher resolution colour stream. A virtual keyboard interface is then rendered into the output video stream, using the hand mask to draw around the outline of the hand so that it appears to exist on a plane of depth above that of the virtual keyboard. When the fingertip position intersects a virtual key from the keyboard UI, that key's letter is added to a character buffer. Once the user has finished a word swipe gesture, a string matching algorithm is used to resolve the set of intersected characters in the buffer to a valid, dictionary word. The system employs basic gesture recognition techniques to provide 'on' and 'off' gestures used to begin and end a swipe, a backspace gesture, and a gesture to select between candidate words returned from the string matching function.

A user study was carried out which tested the system's performance based on text entry speed and error rates, and to determine how users responded to the system in terms of learnability, enjoyability and usability.

It was found that, while the system is hampered by poor text entry speeds and relatively high typing error rates, it provides a potential foundation for future gestural interfaces in that several aspects of the system appear to be promising techniques for gesture-based input. Despite finding the system difficult to use due to its poor reliability and susceptibility to errors, users generally enjoyed the novel aspects of the interface and found its interface both intuitive and learnable.

ABSTRACT

Thanks to improved depth sensing hardware becoming inexpensive and widely available in recent years, gesture-based interfaces now represent a feasible platform for human-computer interaction(HCI). This body of work presents my research and development of a touchless, gesture-based keyboard with an augmented reality interface. I performed a small user study to assess the system's performance and observe how users responded to the system as a new model of computer interaction.

The system combines aspects of the traditional QWERTY keyboard with more modern, gesture-based approaches to typing. To enter text, users perform a swipe gesture in which their index finger traces over the letters of their intended word in one continuous motion. It is specifically intended for health-care environments, where the cross contamination of hands via touchscreens on public computer terminals is a large source of infection spread, and there is therefore a realistic application for touchless typing.

The user study conducted has shown that, while the system cannot yet compete with the performance of physical keyboards in terms of speed and accuracy, it provides a viable framework for mid-air text entry in which users do not need to touch any physical surfaces and also demonstrates the suitability of augmented reality as an interface paradigm for gestural interface systems.

Contents

Acknowledgements

I would like to thank my supervisor Dr. Gerard Lacey for conceiving the initial concept, and for his guidance over the course of my dissertation. I would also like to thank Dr. Jonathan Ruttle for his help in setting up the SDK, and his advice with regards to image processing problems throughout the year. Further thanks to the whole SureWash team for accommodating my work. Finally, I would like to thank my friends and family for their support!

Chapter 1

Introduction

This dissertation explores the design and development of an augmented reality keyboard which provides users with a method of entering text based on simple finger swiping gestures. The system was implemented with a computer vision-based approach, making use of an RGBD camera and a generic display monitor to create an immersive, touchless text entry platform.

My dissertation research has been conducted in partnership with SureWash. SureWash is a Trinity spin-out company which develops software systems for the healthcare industry. SureWash's systems are primarily centered around the use of computer vision and machine learning to automate the training of hospital staff in correct hand washing technique. SureWash's hand hygiene training systems are deployed in hospitals throughout Ireland, Britain and the United States and already make use of gesture recognition to provide technique assessment. SureWash are interested in the potential use of touchless interaction in order to improve hand hygiene standards, and provide an existing hospital kiosk system onto which my virtual keyboard can be deployed. Through my correspondence with SureWash I benefit from having a physical set-up which consists of all of the necessary hardware components required to deploy my system as well as a pre-established network of cooperating hospitals in which my keyboard could reach an active userbase who are already familiar, on some level, with using a hand gesture recognition system.

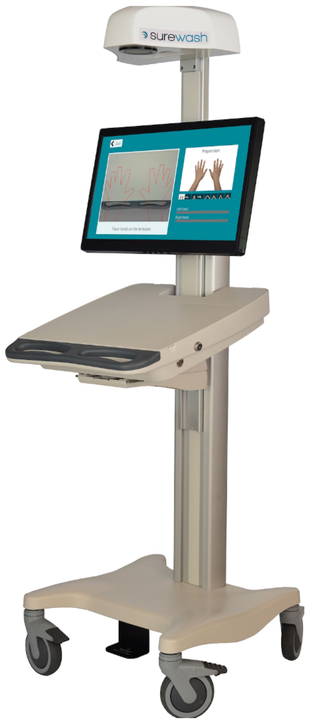


Figure 1.1: A typical SureWash training unit.

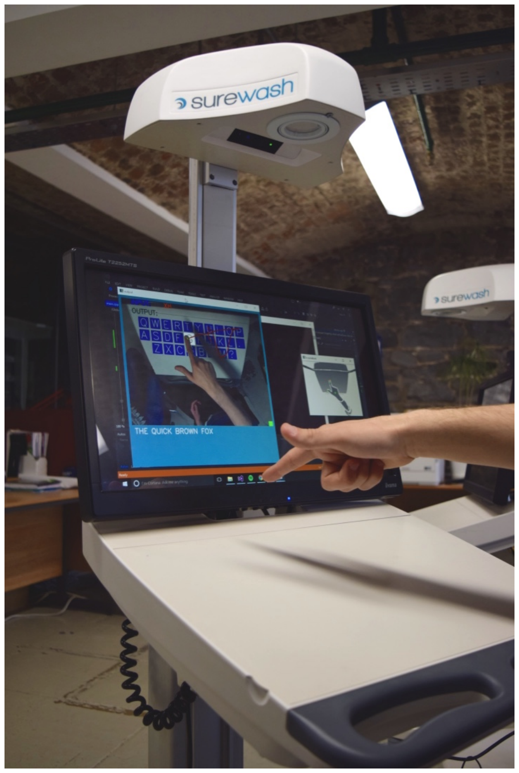


Figure 1.2: The keyboard system running on SureWash.

This chapter presents a description of the motivations for the work, outlines the research objectives of this project and provides an overview of the different sections of this paper.

1.1 Motivation

Ever since the development of the first personal computers in the 1970's, hardware and software capabilities have evolved massively with each successive generation of computer systems. Many core conceptual aspects of the computer, such as the operating system, the user interface, the processor design and storage architecture, have evolved significantly from their early counterparts to the refined, optimized components we have today. However, one facet of the personal computer which has remained largely unchanged, for the past four decades is the way in which we interact with the computer. This field of computer research which deals with how we interact with computers is known as human-computer interaction(HCI).

Between the 1940's and 1960's, prior to the introduction of the personal computer, punch card systems were used to input data into large, primitive computer systems. [1] Following this era, the keyboard was introduced, allowing users to interact with command line interfaces using two-handed typing. In 1973, the Xerox Alto was developed. Although it was never made commercially available, this was the first model of computer to use mouse-driven input. The Alto was a direct influence on Apple's Lisa and Macintosh systems which first introduced the mouse and keyboard interface that we are familiar with today to large-scale commercial audiences.

From this point until the present, the mouse and keyboard interface paradigm, known sometimes as WIMP(windows, icons, menus, pointers), has remained the same in its operation. More recently, as smartphones and tablet devices have become ubiquitous, new human-computer interaction techniques have been widely adopted among portable devices, most notably multi-touch screens. However, the large-scale usage of touch screens in the mobile paradigm has not been replicated in the personal computer paradigm. Nearly all commercially available personal computers and laptops over the past 40 years have used the combination of mouse and keyboard as their primary interface for user input. WIMP has been immensely popular among users due to ease of use and familiarity. While research has produced cutting-edge alternatives HCI models with potential benefits, such as speech-based interfaces and eye-tracking interfaces[2], the paradigm has persisted. However, the WIMP paradigm has been shown to have its limitations.

The Shortcomings of WIMP

In the digital age, as the number of computers(PCs, ATM machines, interactive kiosk systems) and smart devices that we interact with continues to grow, so too does the demand for modular, self-contained devices. The reliance of computers upon extra mechanical HCI controllers such as mice and keyboards is far from the optimal solution in terms of design, usage and installation. Additionally, as GUI's continue to evolve, in situations where a user is interacting with 3D objects, the mouse & keyboard system is limited by the fact that it is bound to 2 degrees of freedom, and cannot truly emulate 3D space. [3]

Furthermore, the concept of the mouse pointer came about as an attempt to virtually mimic the way that we humans use their hands to interact by pointing and reach out toward physical objects, at a time when we did not have the technological capabilities to integrate our actual hands into the interface. Hence, the mouse is ultimately a poor substitute for the hand. These observations indicate that, while the mouse and keyboard interface model has persisted as an effective and popular way to interact with computers, there is room for improvement, and a potential a new paradigm of human-computer interaction to emerge.

The Need for Touchless Interaction

In addition to the perceived drawbacks of the traditional mouse/keyboard model, there are also environments and use cases in which there is a high need for interfaces which are touchless. An example is in hospitals and healthcare environments, where public computer systems intended for shared use act as hot-spots for cross-contamination of hands and subsequent spread of infections among the facility's staff, patients and visitors. A gesture-based interface in which a user can input data in a hands-free way presents a solution to this requirement.

Each year, in the USA alone, healthcare acquired infections(HCAIs) kill over 98,000 people. A patient admitted to a hospital in the United States has a 5% chance of contracting a HCAI, costing over $35 billion dollars per year.[4] Studies have shown hand hygiene to be a major cause of HCAIs. Modern hospitals are now built with interactive kiosk terminals which allow patients, staff and visitors to perform tasks such as check-ins, patient record searches, questionnaires and help queries. The majority of these computer stations use multi-touch screens. [5] However, research has shown that use of such touchscreens in hospitals increases the risk of spreading infections among patients, visitors and staff, as these shared touchscreens can become cross-contaminated and infect anyone who comes in contact with them. [6]

At present, the recommended solution to prevent this spread of harmful microbes is to regularly clean and disinfect these screens. However, it's clear that a more refined solution could be developed using gestural interfaces.The use of a touchless keyboard system in this scenario is an ideal solution for an infection-free text entry method on hospital kiosk. It could be the starting point for additional gesture-based HCI control mechanisms, such as in-browser touchless navigation. Furthermore, the data entry requirements for these in-hospital computer kiosks are usually limited - only small amounts of text/data are inputted in one instance. This consideration may influence user adoption also, since test subjects may be more open to trying a new, unfamiliar HCI controller for use cases of minimal data entry.

1.2 Project Objectives

The desired outcome of my thesis is to develop a touchless, gesture-controlled keyboard interface using a 3D camera and computer vision. I aim to take an augmented reality-based approach to the design of its GUI in order to create an immersive, real-time interface which fuses the users real-world actions with virtual graphical elements. I intend to study the user response to my system in order to determine its strengths and weaknesses. I also hope to gain an insight into the suitability of augmented reality as a GUI approach for systems of this nature, the feasibility of touchless typing systems, and to determine whether the combination of approaches used can help this system overcome the obstacles to user adoption which past gestural interfaces have encountered.

1.3 Dissertation Overview

Chapter 2 provides an overview of the field, its history, and the related work that has been produced in recent years which is relevant to my own work. In Chapter 3, some of the core background concepts to my thesis are discussed. Chapter 4 provides a look into the design of the system, while Chapter 5 cover its implementation. Finally, Chapter 6 discusses the user tests that were performed, how users responded to the interface, and future work which expand upon my findings.

Chapter 2

Related Work

Evolution of Gesture-Based HCI's

"Gesture is the richest possible digital input that we, as humans, can deliver".
This quote from John Underkoffler, a researcher and UI designer from MIT, underpins much of the motivation and inspiration for my work. Underkoffler is most well-known for being the man behind the iconic gesture-controlled interface depicted in the 2002 film "Minority Report". The futuristic UI concept has been highly influential in the field of natural interface design ever since its emergence into popular culture at the turn of the century, and for many people, is the first thing that comes to mind when they think about gestural interfaces. However, gesture-based control is a much older discipline in which research has been taking place for decades.

- One of the earliest notable examples of a gesture-controlled interface was the "Put-That-There" system, developed by RA Bolt in 1980. [7] "Put-That-There" was a multimodal system, fusing speech and hand gestures to let users prompt events on a screen by combining voice commands with pointing gestures. At the time of its development, it was state-of-the-art and regarded as an impressive feat of technical innovation.

  For capturing hand gestures, the sensor-based system utilized a magnetic device worn by users on their wrist. Speech events were captured with a microphone and speech recognizer. Through use of the two sensor types, users were able to construct and arrange simple shapes.

- In a 1987 paper, Thomas G. Zimmerman presented the DataGlove. This glove-based device, which was released to the market by VPL Research, became the first commercially available gesture interface. The DataGlove used an optical flex sensor to allow for the detection of a user's hand position, orientation and the bend in their fingers. At the time of release, the DataGlove boasted hand tracking with six degrees of freedom in real-time. However, its main drawback was that it was mainly suited to capturing simple hand gestures, the device was reported to not be sufficiently accurate at registering more complex gestures which involved finer manipulations of the hand. The DataGlove was also limited by its capture speed, which restricted it from successfully recognizing swift hand motions.

- In 1991, Pierre Wellner of Xerox Parc published a paper which documented the development of the DigitalDesk. This was a notable early example of the composition of computer generated elements onto a real-world display. It was also one of the earliest successful gesture interfaces to use computer vision. The system consisted of a projector and camera which were mounted above a physical tabletop surface. A graphical interface was projected onto the physical surface and the camera could capture gestures made by a user relative to these graphical elements and modify the display in response to the user's actions.

  Wellner was interest in digital documents, which, at the time, were expected by many to replace paper documents in the workplace. Instead of focusing his work solely on digital files, Wellner was interested in the fusion of physical paper documents and electronic documents, citing the fact that "paper has properties that people just cannot seem to give up" which make it unlikely to be made entirely obsolete by digital alternatives.

  The DigitalDesk featured several different applications which were based around virtual interaction with graphical objects being projected onto physical documents. One such application was a calculator in which a user could place a sheet of paper onto the tabletop and use their fingers select different values on the sheet. The DigitalDesk's camera would capture this data so that it could be processed by the underlying vision system which could recognise the specified values to perform additional calculations and then redisplay. The system also featured a translation application which allowed a user to point at unknown words(for example, in another language) which the underlying system could recognize and translate to English for re-display via the projected interface. Another feature was a paint-based application in which a user could draw onto the physical paper using a pen/pencil, and then select their drawing and copy/paste it using the projector-based digital interface.

This DigitalDesk was a landmark in AR systems. It was a significant early demonstration of how the physical and digital world could be fused in a practical way. It illustrated a way to give electronic properties to physical documents and vice versa, letting users retain the inherent advantages of each approach.

- In 1994, another early hand gesture recognition system which used computer vision was developed by James Davis and Mubarak Shah. [8]. This system required users to wear a glove with fingertip markers. A camera was used to capture the video frames which were then processed to identify hand gestures. The system performed impressively given the available hardware at the time, and was a precursor for much work in the field of vision-based touchless HCI's.

The Field Today

In recent years, hardware has finally reached a point where a vision-based solution can be implemented with sufficient performance at a low cost. Over the past 5 years, a number algorithmic approaches to vision-based gesture recognition have been proposed and many different types of gestural interfaces have also been developed, some systems use the hand to simulate a mouse cursor and others forgo the traditional cursor-based scheme for new gestural navigation schemes. Focusing purely on vision-based gesture-controlled keyboards, some notable implementations are documented here:

- In a 2010 paper entitled "Anywhere Touchtyping", Mujibiya et al. detailed the development of a virtual keyboard which allows users to type by tapping on an arbitrary surface. [9] The system takes the unique approach of using a DLP projector to illuminate a pattern onto the surface and then using a phase shift algorithm for depth detection. The project makes use of computer vision techniques to process the captured frames and identify when a user has touched the surface and made a keystroke. Though this system shares some conceptual similarities with my own work, my project expands upon several of its core aspects and contains differences in its implementation. Rather than a DLP projector, my system uses a 3D camera to capture depth data and estimate key presses. Furthermore, my approach to the interface is also fundamentally different, in addition to it being completely touchless.

- Another notable virtual keyboard system is the Gestairboard, a virtual keyboard system developed by researchers at the Technical University of Munich in 2013. [10] The Gestairboard was the first touch-typing keyboard to use the Microsoft Kinect, demonstrating that virtual keyboard systems could be implemented on inexpensive, off-the-shelf hardware. The Gestairboard has a simple UI which shows a virtual keyboard overlay on the screen and allows users to input text data using the traditional 10-finger typing system. Though the system functioned as a proof of concept for a touchless, gesture-based keyboard, its usage proved to be somewhat difficult for test subjects. The researchers felt that it could be improved on in terms of speed, error-rate and visualization. My project will investigate alternative typing schemes, mainly the word gesture key approach outlined in Chapter 2, which aims to improve typing speed and provide a text-entry approach which is more appropriate for touchless typing. I aim to address the authors' recommendation for improved visualization through the development of an interactive AR overlay.

- One existing touchless keyboard that implements a swiping text entry algorithm is Vulture. Vulture is a "mid-air gesture keyboard" which allows users to perform swift data entry by tracing words. The authors of the paper aimed to investigate the speed of text entry that could be achieved through mid-air interaction. To operate the system, a user wears a special glove covered with reflective markers and forms a pinch gesture with their thumb and index finger before swiping across the letters that make up words. The Vulture system performs well and demonstrates that word gesture keyboards lend themselves well to a touchless, text-entry interface. However, the system is designed to use a data-glove sensor on the operating user's hand. Also, the interface is completely virtual, with the user's fingers controlling an on-screen cursor. I intend to use computer vision in my implementation to eliminate the need for any wearable parts to the system. The only required component should be the users hand. Furthermore, I aim to base the system's GUI around the real-time camera feed, making the user's actual hand a core part of the display, and to study how this affects usability and immersion. Rather than providing a point of control for a mouse cursor, the users fingertip becomes the cursor itself.

Usability

In addition to the above experimental interfaces, a number of other hand gesture-based interfaces have emerged over the past few years. However, at the time of writing, no single interface has taken off as a largely deployed, commercial gestural interface. Many usability studies have been carried out which analyze user reactions to gesture controlled interface projects. Several of these studies were consulted during the design process of this system in order to get an idea of common user perceptions to this kind of system and to identify the common barriers to user adoption which arise.

In particular, three user studies were examined. Though none of the studies were conducted using a HCI device that is exactly analogous to a touchless keyboard, all used depth sensing cameras(mostly the Kinect or the Leap) and focused on applying hand gestures to interact with UI elements. One such study, a thesis paper entitled

'Usability Evaluation of the Kinect in Aiding Surgeon-Computer Interaction', found that surgeons reacted positively to a gesture-based system in the operating room.[11] The study was carried out in early 2013 and used a Kinect to allow surgeon's to view and manipulate medical imaging records through hand gestures. It concluded that surgeons rated the usability of the system highly, and that future improvements in hardware could only further enhance the system's performance and user impressions. This paper stood out as an example of the successful deployment of an experimental gestural interface within a healthcare environment, the same domain for which my system is intended.

Another study from 2011 entitled *'Using Commodity Visual Gesture Recognition Technology to Replace or to Augment Touch Interfaces'* which compared a Kinect-based gesture system with a touchscreen noted that the sample user group preferring to use the touch screen in all cases. [12] However, one particular weakness of this system, which could be improved upon, was that users found the full-hand gestures involved to be unintuitive and non-obvious. The authors were limited by technological constraints, but posited that in the future, more sophisticated, finger-based gestures could enhance usability.

A third study, from 2013 entitled *'Design and Usability Analysis of Gesture-based Control for Common Desktop Tasks'* showed promising results. This study tested users who manipulated UI elements with both arm gestures and finger gestures in place of using a mouse. The study used the Kinect along with the OpenNI, NITE, and OpenCV libraries. It concluded that finger gestures were received more favourably than arm gestures, as they felt more natural to users and caused less fatigue.[13] However, users did not find finger gestures to be easier or quicker than arm gestures, but found them less fatiguing and more natural.

Adoption of Gesture-Based Interfaces

The referenced usability studies make it evident that when hardware and software perform to an appropriate standard, users generally respond positively, especially so to more minimal, finger-based gestures. This observation influenced the focus on fingers as a control vector for the presented AR keyboard system. Conversely, a replacement for the mouse & keyboard has still not been found. There are a number of possible answers to the question of why hand gesture-based interfaces have not yet been commercially successful.

1. Firstly, it's clear that until recently, the technology required to seamlessly and accurately implement these systems was not available. Only in the past 4 to 5 years has hardware such as depth sense cameras reached sufficient levels of resolution, frame-rate and performance to enable us to build efficient gesture based systems.

2. Secondly, the unfamiliarity of the common user with gesture controlled systems could be a factor in public resistance. Generations have grown up with the physical mouse & keyboard being the constant standard and a new, virtual mechanism may naturally meet resistance.

3. A lack of tactile feedback compared to a mechanical input system or touchscreen interface may be another aspect of touchless interaction that causes unsatisfactory user responses.

4. While real-time gesture recognition can now be implemented to perform efficiently and reliably, studies have shown that the speed with which users enter text is still slower than with the mechanical keyboards that they are used to. [10] It may therefore be necessary to re-think the text entry approach taken, as the traditional QWERTY mechanism may not be optimal for a touchless keyboard. An alternative text-entry approach may provide the solution to this issue.

Chapter 3

Background

3.1 Gesture-Based Interfaces

One area of research that shows promise as a new paradigm for HCI is in hand gesture-based interfaces. Humans naturally use hand gestures for communication and expression, and the ability to apply this inherent human instinct to our interaction with computer systems could offer innumerable benefits in terms of speed, learnability and ease of use. [14] [15] In place of the cumbersome mouse and keyboard combination, a new generation of computers that use a gesture-based HCI solution would allow us to interact with these systems in a manner that appeals to our primal intuition for communicating our intentions through body language.

A *gesture* is defined to be any physical movement of a person's hand, arm, face or body for the purpose of conveying meaningful information.[16] A gesture-based interface is a user interface which recognizes a user's gestures as input and uses them to provide control to a device or system. Although systems have been developed which allow for the recognition of facial gestures, eye-tracking and even full-body gestures, my work is concerned with movements of the arm, hand, and fingers, and devising a way to use these to control a virtual QWERTY keyboard.

In the field of gesture controlled human-computer interaction, two main classes of interface have emerged: sensor-based systems, and vision-based systems.

1. **Sensor-based methods** utilize one or more different sensors, such as accelerometers, magnetic sensors and optical sensors, to capture gesture data.

An example of this can be seen in the DataGlove, a HCI controller which the user wears on their hand that uses motion sensors to track finger positions and orientations and translate these into device control signals. [17]

2. **Vision-based methods** use one or many cameras to capture the user's hand movements and then apply computer vision and gestured recognition techniques to process the captured data in order to determine what gestures are being employed by the user.

This dissertation is solely based around the approach that uses computer vision which is preferable to data gloves in terms of usability since it does not require the user to wear any additional components, and can be implemented using inexpensive, easily available 3d camera hardware. As outlined by [16], an effective vision-based gesture recognition system must be robust enough to handle noisy and incomplete data due to variations in image illumination and occlusion, and it must also be computationally efficient so that they can operate in real-time.

3.2 Augmented Reality

Augmented reality(AR) is a user-interface paradigm in which virtual objects are blended with a user's view of the real world. In AR interfaces, computer-generated objects are superimposed into a video feed in a smooth, immersive way. AR has become increasingly relevant in recent years, as many popular mobile app developers have begun to utilize AR to create interactive applications for entertainment, gaming and educational purposes. Augmented Reality can be contrasted with Virtual Reality, which is based around the construction of a virtual world which users can interact with. The distinction between the two concepts is that AR involves the user interacting with virtual objects in the real world, while in VR, the user is removed from the real world. AR aims to enhance a user's real-world surroundings, rather than simulate a new environment.

Though researchers have been exploring the integration of virtual and real-world objects in some form since the 1970s, Augmented Reality itself is a relatively young field, which only became clearly defined in the 1990s. In a 1997 survey, Ronald Azuma published a survey which characterized the field and its requirements. [18] For a system to qualify as an AR system, it should: blend real and virtual objects, run in real-time, and correctly register the virtual objects with those of the real-world. Registration is the process of accurately aligning real and virtual elements in relation to one another. It it necessary in order to create the impression that the different elements exist in the same environment.

One popular use case of AR is in head-mounted display systems(HMDs), such as Google Glass, and the HTC Vive, in which users wear a head-mounted hardware device which consists of a processor, a camera and a small display. AR-enabled applications running on a HMD can utilize the real-time camera feed to augment graphical objects into the wearer's field of view. Another approach use of AR is in smartphones and tablet devices, where the camera in the handheld device provides the real-world view. And finally, as can be seen from my approach, AR may be executed in any system that combines a monitor with a camera. In my case, the combination of a desktop monitor with an RGBD camera provides all the necessary hardware.

In order to solve the registration problem and align real and virtual objects, information must be derived about the location/position of the area of interest in the real-time video. For example, if a 3D graphical object is being rendered onto a real-world tabletop surface in a video, the location and orientation of the surface needs to be determined before its relationship with a new 3D object can be established. To solve this problem, there are 2 main approaches: marker-based AR, and markerless AR. In marker-based augmented reality, systems rely on fiducial markers (such as a piece of cardboard of known size, or a QR code) to operate. By calibrating the system's camera and using known properties of the marker such as its size and shape, information about the surface on which it is placed can be derived. Marker-less AR, on the other hand, uses computer vision techniques to recognize an object's features which can be used to inform the blending process.

The goal of AR is to enhance a user's perception of the real world. In a touchless system, without any sort of physical interface the user will rely almost entirely on visual cues when interacting with the system. AR assists usability and offers a way to provide an interface that is intuitive and engaging despite having no physical properties in the real world. My graphical interface utilizes AR by projecting a virtual keyboard graphic into the scene of a live camera stream. The virtual keyboard is rendered at a specific plane on the z axis so that it 'floats' in mid-air, above any physical surfaces. The keyboard object is the sole component of the user interface, acting as the user's frame of reference between the real world and virtual. Without any haptic or tactile feedback, the development of an immersive, intuitive virtual keyboard interface is paramount to the success and usability of this system.

3.3 Word Gesture Keyboards

One issue faced by existing implementations of touchless keyboards and other gesture-based interfaces is the gap in achievable typing speeds between touchless interfaces and those of physical keyboards. With the emergence of mobile and tablet devices in recent years, alternative text entry approaches have become increasingly popular as a way to get around the limited typing speeds achievable on devices with much smaller keyboard interfaces. I intended to investigate the use of one of the most popular alternative text-entry methods, one that is more optimized for gestural typing: the word gesture keyboard.

Word gesture keyboards(WGKs) are based around the idea of letting users enter text by tracing patterns across the screen instead of pressing down on individual keys. The user enters a word by swiping their finger, or a stylus, over the letters of their intended word in one continuous stroke.

One of the earliest implementations of a word gesture keyboard was IBM's SHARK. SHARK, or Shorthand Aided Rapid Keyboarding, is an approach to speed-writing that was developed by Per Ola Kristensson and Shumin Zhai in 2003. The text entry paradigm was initially developed for stylus keyboards, with the creators expressing their intention for their input model to facilitate users gradually progressing from visually-guided gestured to recall-based gestures. [19]

In 2012, the authors remodeled SHARK to be more suitable for mobile platforms, replacing the stylus with the finger.[20] This new WGK implementation, known as ShapeWriter, was developed due to dissatisfaction with the typing mechanisms for existing mobile touchscreens which are often tedious and error prone. ShapeWriter instead takes the approach of viewing the letters of a word as that word's geometric code. Each word in a lexicon has its own code on a keyboard layout and to input that word, the user swipes their finger over the keyboard in a single stroke, tracing out a word. The authors compared the process of swiping a word to cursive writing, in which letters are joined in a continuous motion, rather than the one-at-a-time approach which traditional touch typing and print writing share.

ShapeWriter, and similar swipe-based algorithms such as Swype, SwiftKey and Touch-

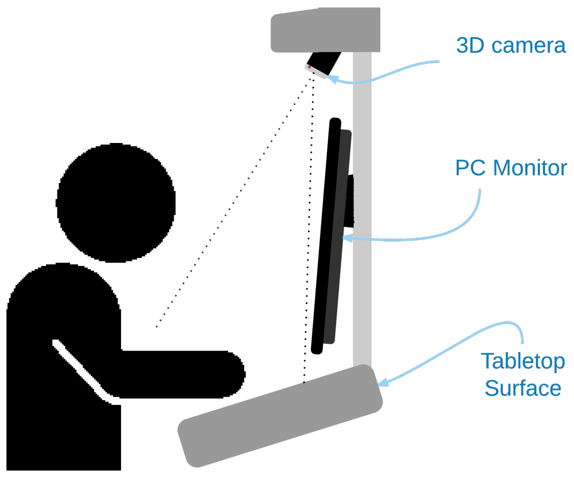
Pal have become popular on mobile platforms in recent years. This approach offers speed and high efficiency and lends itself well to an AR keyboard, as WGKs allow users to enter text using intuitive gestures. Furthermore, to my knowledge, no touch-less keyboard has been implemented so far which combines both an AR interface with the word gesture mechanism.

Chapter 4

Concept & Design

4.1 System Concept

The physical setup of the system consists of a display monitor, a depth sensing camera mounted above the monitor, and a flat tabletop surface that is positioned at a >90° angle to the monitor. In order to operate the system, the user must hold out their hand above the surface at which the camera is pointed. Note that the surface is tilted and not perfectly parallel to prevent users from resting objects upon it, which would interfere with depth-based segmentation. The adjacent screen displays the camera's colour feed in real-time as the user interacts with it. The user should never actually touch the tabletop surface, but rather it is intended as a visual frame of reference for their touch-less interaction with the system.



The system will overlay a virtual QWERTY keyboard into the scene with which the user interacts. To type, the user positions their index finger over the first letter of their intended word and forms a hand gesture to initiate the word. They then swipe over all of the letters in their word until the final letter of their intended word is reached, where a new gesture is made to indicate that the swipe is finished.

GUI Design

There are numerous different approaches that could be taken to create a GUI for the system:

- **Optically Projected Keyboards** - One type of GUI which has been used for keyboard applications and other HCIs is the projection keyboard. With this method, the image of a keyboard is projected onto a flat surface, usually using a laser. The user interacts with the keyboard application by tapping the sections of the surface which display the image of a key. An example of this type of keyboard can be seen in Cellulon's Magic Cube, pictured below. The DigitalDesk demonstrates similar functionality with of its projected interface.

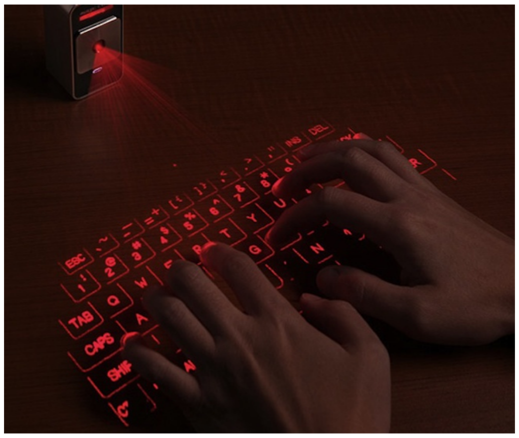


Figure 4.1: Cellulon Magic Cube [21]

- **Virtual Keyboard** - Another approach that can be taken to the design of a keyboard interface is to completely remove any physical aspect from the interface and have the keyboard represented solely as a software component. In addition to mapping the set of keys as virtual buttons, such keyboards must also have a method of representing the users interaction with the keys. This can be done by using a cursor to represent each finger, reconstructing a virtual hand to represent the user's hand, or in the case of touch screen devices, illuminating the pressed key. Example of this kind of keyboard can be seen in the Gestairboard, as well as the Ipad, pictured below.

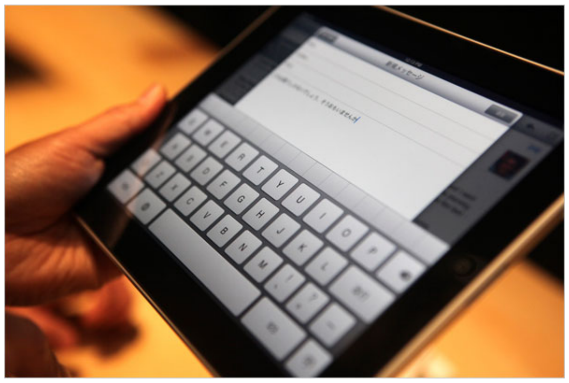
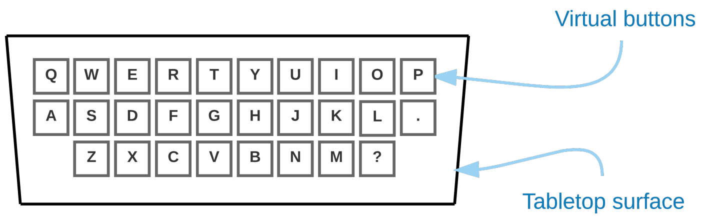


Figure 4.2: Typing on the Ipad's virtual keyboard [22]

- **Augmented Reality Keyboard** - An AR keyboard fuses a video stream of the user's hands with a virtual keyboard interface. The AR approach has advantages in that it doesn't require extra hardware components such as a projector, and also that it should be more intuitive to use than a wholly virtual UI, since it adapts the view of the user's actual hand, rather than simulates. A weakness often faced by virtual approaches which use a reconstructed hand or mouse cursor is that a perfect reproduction/simulation of the user's movements in real-time is very difficult to achieve. There are often finger movements in the graphical hand reconstruction which are inconsistent to the real hand. For example, the hand or cursor may make small movements while the real hand stays still, or vice, causing a perceived lag or lack of responsiveness in the system UI. An AR-based approach eliminates this issue by only simulating the static keyboard interface itself, and rendering it around the user's hands so that they are able to witness the real movement of their hands in real-time.

I opted to employ an AR-based design for the system's GUI. The keyboard would be rendered as a set of keypad buttons drawn over the real-time camera frame. Since the hardware camera is pointed down to the tabletop surface from above, the buttons can be drawn as 2D squares containing that key's corresponding character. For the initial system design, I opted to keep the UI simple and minimalistic, rendering the keyboard only as a grid of floating keys, with no surrounding border UI object. The button grid rendered into the video stream and aligned so that it is positioned on top of the video image of the tabletop surface.



An issue faced by many existing touch-based keyboard systems which employ AR is that the hand does not occlude the virtual keyboard. The keyboard interface is simply drawn over each camera frame without reacting in any way to the presence or location of the actual hand/fingers in each frame, rather using a virtual marker to highlight the fingertip. Figures 4.3 and 4.4 below demonstrate this "over the hand" approach to AR-keyboard design in two existing systems. Rather than floating over the buttons as a real hand would do when it is located nearer to the camera than the keyboard, the hand is seen to be underneath the keyboard.

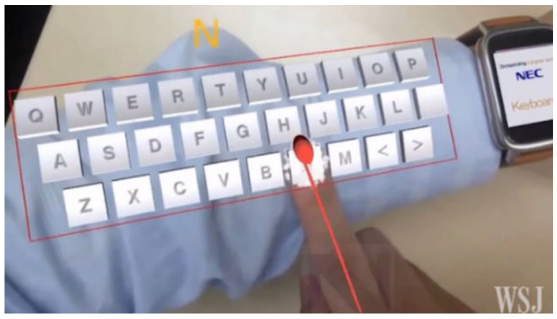


Figure 4.3: NEC's ARmKeypad [23]

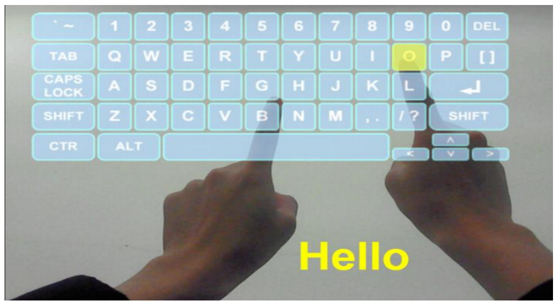


Figure 4.4: KAIST's K-Glass 3 [24]

This "over the hand" style of interface is not optimal for top-down keyboard views as it breaks immersion. In order to accurately simulate the act of a user pressing down upon a button with their finger, the finger should appear closer to the camera than the keyboard. My solution is to create the illusion of the keyboard existing underneath the hand by masking around hand. The purpose of this approach is to augment the real world in a more realistic way and let the user interact seamlessly with the keyboard as if it were a physical keyboard.

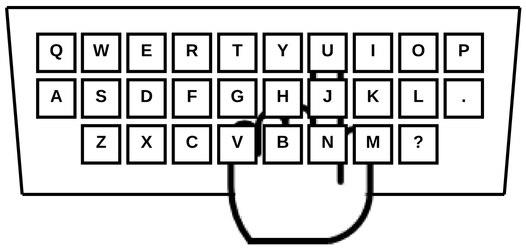


Figure 4.5: Keyboard rendered "Over the hand". This approach does not achieve the desired level of realism.

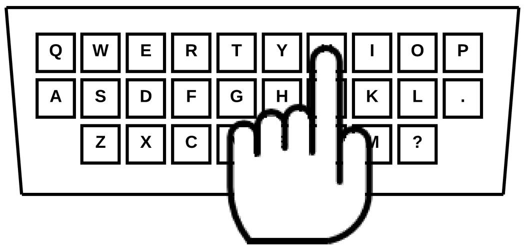


Figure 4.6: My solution: keyboard is rendered "under the hand" by masking the hand from button grid.

Button Sizing Considerations

The size of keypad buttons and the spacing between them is an important aspect of the design to consider, as it has a significant impact upon the system's user friendliness. The ease with which users can select an on-screen target is influenced by the size of the target and its distance from the cursor. From Fitts' Law, we know that the design of a button-based UI should minimize the total distance for the cursor to cover while keeping targets sufficiently large in size. [25] This methodology can be applied to my system by keeping the dimensions of the whole keyboard grid small enough so that the user can swipe from the top left key 'Q' to the bottom right key '?' in a short amount of time, while still allowing the keys to be large enough that accidental mis-selections are uncommon. The overall size of the keyboard grid is a function of both key size and spacing between keys.

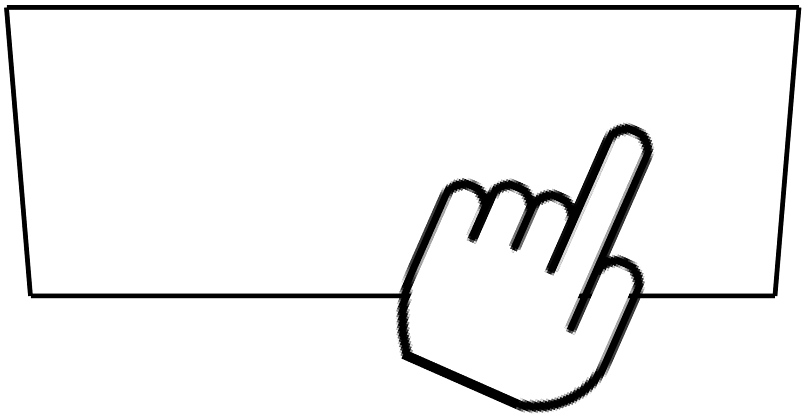
Due to the user's hovering hand, moving freely in 3D space, being the primary control mechanism for the system, users will likely be susceptible to erroneously pressing neighbouring keys around their desired key during swipes. Furthermore, natural jitters and tremors which occur in the motion of the human hand could potentially translate to unintended key presses if the pointing finger drifts over a new key boundary. To try to mitigate errors of this nature, methods were investigated with the intention of improving user performance and making it easier to press the desired key each time.

Existing studies have shown "magnetic" cursors to be an effective mechanism for improving target selection in virtual and freehand gesture-based interaction systems. [26], [27] The magnetic cursor approach applies to systems where the user is controlling a cursor with either a mouse or their hand gestures. A "snap-to-target" effect is implemented which causes the cursor to jump towards the target when it is sufficiently close. Furthermore, once the cursor is positioned over the target, its gain is reduced so that it moves more slowly, effectively making it more difficult for the user-controlled cursor to accidentally drift off the target. This mechanism, while effective for assisting usability, is less suited to an AR-based system where the traditional cursor is replaced by the user's actual finger. To alter the real-time video stream used in the GUI in order to create the illusion that the finger moves more slowly over buttons/targets would hamper the user's perception of the system's responsiveness.

One mechanism which does lend itself to an AR-interface is target expansion. Temporarily expanding the target when the user's pointer finger gets closer allows the advantages of a larger target size to be utilized while not causing an overall increase in the size of the keyboard grid. This feature was implemented into the system so that the nearest key to the finger expands and remains at a larger size until the finger moves toward another key. The active key's colour is also changed, compared to the rest of the keypad buttons.

Output Display

An initial frame from the camera's colour stream, before any graphics have been drawn into the scene, will simply show the user's hand over the flat tabletop surface, from the camera's viewpoint:



After augmenting the virtual keyboard into the frame and masking it around the hand along with applying the cursor expansion and rendering the swipe path of the index finger, the re-displayed frame will be of the form:

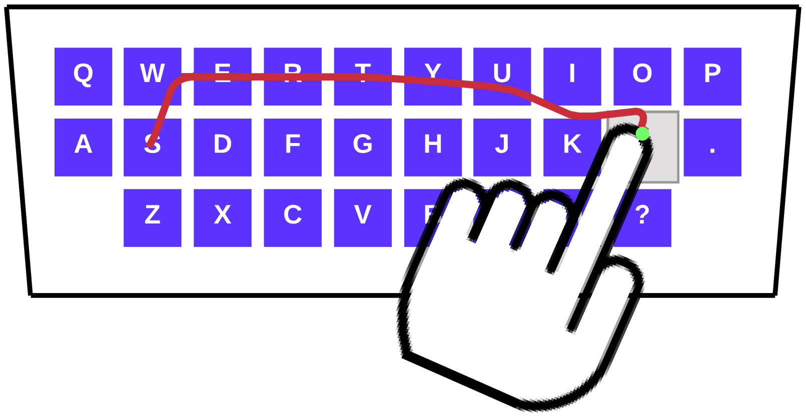


Figure 4.7: Swiping the word "sell"

The swipe path is simply a highlighted polyline which conveys the total path taken by the finger in the current swipe gesture. In addition to expanding the active target key, the fingertip point is highlighted by the system as a small green circle at the tip of the finger.

Audio Feedback

In order to improve usage cues in the absense of tactile feedback, the decision was made to expand the system from a unimodal visual interface into one which employs both visual and audio feedback. Audio feedback is provided by having a click sound emitted from the system's speakers upon each key press. It is a simple, yet effective way to inform the user that a key press was recognized by the system and in the case of a swipe, it notifies the user as to whether their swipe is working and how many keys are being activated by their swipe. The use of key press sounds provides an additional cue to the user and reduces their need to focus solely on the visual elements of the system.

4.2 Camera Hardware

Building machines with stereo vision systems that can capture their surroundings in real-time, in the same way that humans use their eyes, has been a feat which engineers have long strived to achieve. In recent years, advancements in hardware have made this kind of 3D vision system available to almost anyone, due to a surge in the number of different depth sensing cameras that have been released at low costs.

At present, there are a number of leading depth sensing cameras on the market. One of the most popular models of RGBD camera is the Microsoft Kinect. Version 1 of the Microsoft Kinect was first launched in late 2010 as a motion tracking peripheral for the Xbox-360. However, it soon became popular among computer scientists and engineers for experimental use in 3D vision projects. This prompted Microsoft to introduce a PC-compatible version in 2011 along with an SDK for languages such as C# and C++. This first generation used structured light to infer depth information and excelled at skeletal and face tracking, but lacked a sufficiently high resolution to easily distinguish fingers on a hand. The second generation of Kinect was released in 2014. Featuring an RGB camera, an infrared camera and a depth processor, it has a higher resolution and more accurate depth sensing, as well as improved hand gesture tracking. It has subsequently become the leading device for real-time tracking of human bodies.

Another popular depth sensor is the DepthSense from SoftKinetic, which is used in their DS325 camera. The DS325 provides both a colour camera and depth sensor. The device's depth sensor uses time-of-flight technology for depth calculation. In time-of-flight sensors, the sensor estimates the depth of objects in the camera's view by measuring the time taken for infrared light to be emitted, bounce off the object surface and return to the sensor. The DS325 provides colour video resolution at 640x480 and depth resolution of 320x240, with rates of up to 60 frames per second. The DS325 sensor can track hands and fingers within a range of 15cm to 1 metre.

For the implementation of this project the DS325 was used as it is more optimized for shorter distances and fares better for subtle changes in user hand gestures. Time-of-flight sensors are also less sensitive to noise than other depth acquisition technologies.

4.3 Software

4.3.1 DepthSense SDK

For the implementation of this system, I used SoftKinetic's DepthSense SDK to programmatically capture depth and colour data from the camera used in my project setup. The DepthSense SDK is a programming interface for the DS325 camera which provides access to depth maps, RGB data, and confidence data. It also provides functions which map the pixels of higher resolution colour frames to corresponding low resolution depth images, so that the user can find the RGB value of a pixel in the depth map.

Chapter 5

Methodology

This chapter details the implementation of the specified touchless keyboard system. Section 5.1 provides an overview of the system and then main components of the system pipeline, while subsequent sections provide more in-depth descriptions of the steps taken during the development of each stage of the system.

5.1 Implementation Overview

The system can be broken down into three logical components:

- The **image processing** system which is responsible for performing all of the computer vision-based operations required in order to analyze each frame captured by each of the DS325's colour and depth streams. In the image processing stage, OpenCV [28] is used in order to determine if hands are present in the camera viewpoint, segment the hand shape and detect the x,y,z position of any fingertips found, and determining the user's gesture based on the fingers found. All of this must be performed in real time. If fingertips have been successfully detected in the scene, this stage is also responsible for building up the path traced by the finger(based on x,y,z coordinates) over the button boundaries of the virtual keyboard GUI and producing a string containing all the keyboard characters which a user has swiped over in each swipe gesture.

- The **string translation stage** performs the approximate string matching required to take the input string of swiped characters and determine the set of words which the user most likely intended to produce, returning a set dictionary words which are ranked based on this likelihood.

- The **display** stage involves drawing an AR-style interface over each camera frame in real-time. This stage involves rendering a graphical keyboard comprising a set of virtual buttons which models a typical QWERTY keyboard. In order to blend the virtual elements of this keyboard with the real-world data captured in the camera's colour stream, a mask of the hand is computed and removed from the keyboard UI so that the illusion is created of the hand floating above the virtual interface. This stage is also concerned with drawing the output 'swipe path' which depicts the trail of keys which the user has selected during each swipe, and also the final display of each word typed by the user.

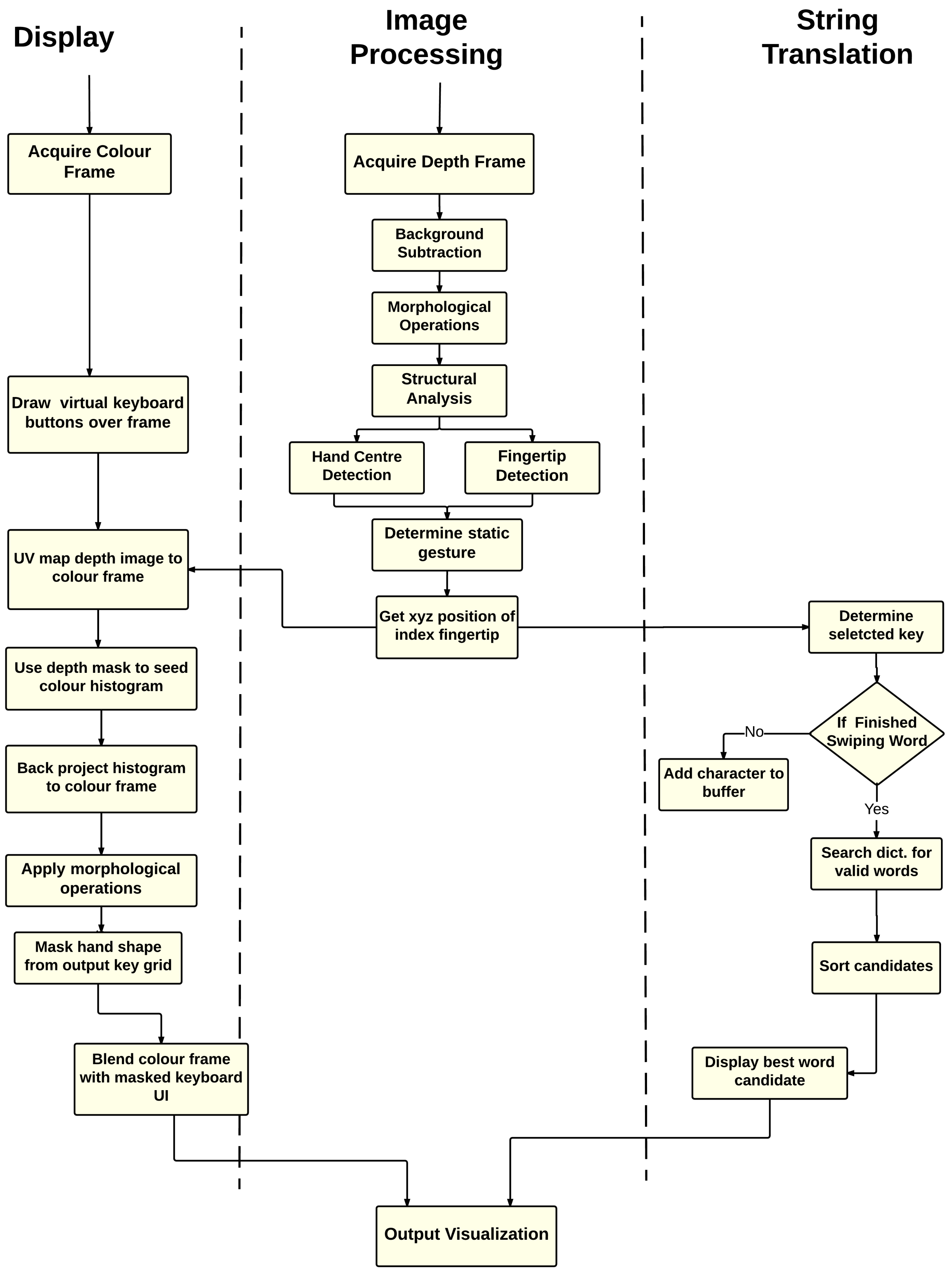

Display
Image Processing
String Translation
Acquire Colour Frame
Acquire Depth Frame
Background Subtraction
Morphological Operations
Structural Analysis
Hand Centre Detection
Fingertip Detection
Draw virtual keyboard buttons over frame
Determine static gesture
UV map depth image to colour frame
Get xyz position of index fingertip
Determine seletcted key
Use depth mask to seed colour histogram
If Finished Swiping Word
No
Add character to buffer
Back project histogram to colour frame
Yes
Search dict. for valid words
Apply morphological operations
Mask hand shape from output key grid
Sort candidates
Blend colour frame with masked keyboard UI
Display best word candidate
Output Visualization

Figure 5.1: A flowchart for the operation of the overall system.

5.2 Image Processing

OpenCV is the library used to provide functions required for many of the image processing operations performed in my code, along with the DepthSense SDK which is used to grab incoming frames from the hardware camera device. For colour frame captured by the camera, it is matched to the corresponding depth frame based on its timestamp. Each frame is stored in an OpenCV Mat object. Frames captured by the depth camera are stored in single channel(CV_8UC1) Mat structures of size 320x240, while colour frames are stored in 3-channel(CV_8UC3) Mat structures of size 640x480.

5.2.1 Hand Segmentation

To segment the user's hand, background subtraction is used. Since the system's camera hardware is fixed in place so that the camera points down towards the tabletop surface, the captured viewpoint will contain a static background. Therefore the approach is suitable for segmentation of the hand. Depth data was used to segment the hand, because it is unaffected by changes in illumination and shadows. These factors can influence the accuracy of a colour-based background model. When interacting with the application, the user's hand will be held above the tabletop, within the camera's field of view. Hence, the hand will be at a higher depth level than that of the surface, and it can be easily segmented based on depth. The confidence map for each depth frame is used to disregard any pixels whose confidence value is low during the model computation.

Background subtraction is a segmentation technique used to separate foreground structures from their background in a sequence of input video frames. This technique involves computing a background model of the scene based on frames in which there are no external objects in the camera's field of view. This model establishes which pixels belong to a scene's background. Foreground objects can then be segmented by calculating the difference between the background model and each subsequent input frame. To segment the hand in the output in binary depth image, set all pixels(i,j) to 1 where:

$$f_k(i,j) - b(i,j) > T$$

where f_k represents the current frame, b represents the background model, and T represents a threshold value. [29]

While there are many approaches which can be taken to construct a background model, for this system, a running average method was chosen. This has the advantage of being able to effectively reduce noise occurring in depth frames. Using a running average constructs the background model by averaging a set of frames. At each pixel, the mean is computee for the same pixel coordinate in the previous n frames. Rather than inducing a high memory cost by storing the previous n frames, as is done in the traditional mean filtering technique, the running average approach approximates the mean using a learning rate α. The learning rate α is a value between 0 and 1 which gives weightings to different frames, with more recent frames typically having a higher weighting. [29]

$$b_n(i,j) = (1-\alpha)b_{n-1}(i,j) + \alpha f_{n-1}(i,j)$$

The background model is computed upon initialization of the application. It requires that the tabletop surface is clear and that no hands are present in the scene for the first n frames. The running average-based background model initialization is implemented in OpenCV using the accumulateWeighted() function. Figures 5.2 - 5.4 below show the effect of subtracting the current depth frame from the depth background model, and producing a resultant accurate hand mask. A subsequent opening morphological operation is then performed. This consists of an erosion followed by a dilation. Its purpose is to remove small islands of noise while retaining the shape of larger structures.

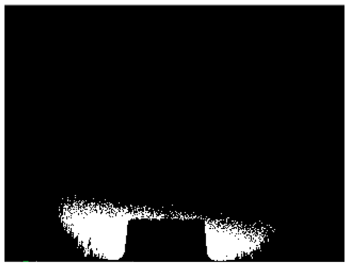


Figure 5.2: Background model

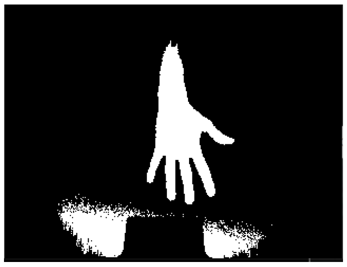


Figure 5.3: Current frame

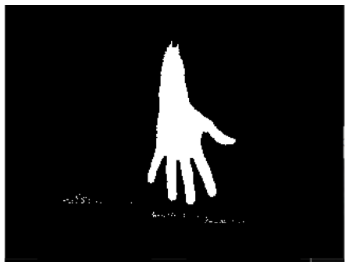


Figure 5.4: Frame difference

5.2.2 Structural Analysis

Following the segmentation of any foreground objects from the background depth scene, an analysis of the structure is performed in order to confirm that it is likely to be a hand, and to facilitate fingertip detection and hand centre localization procedures.

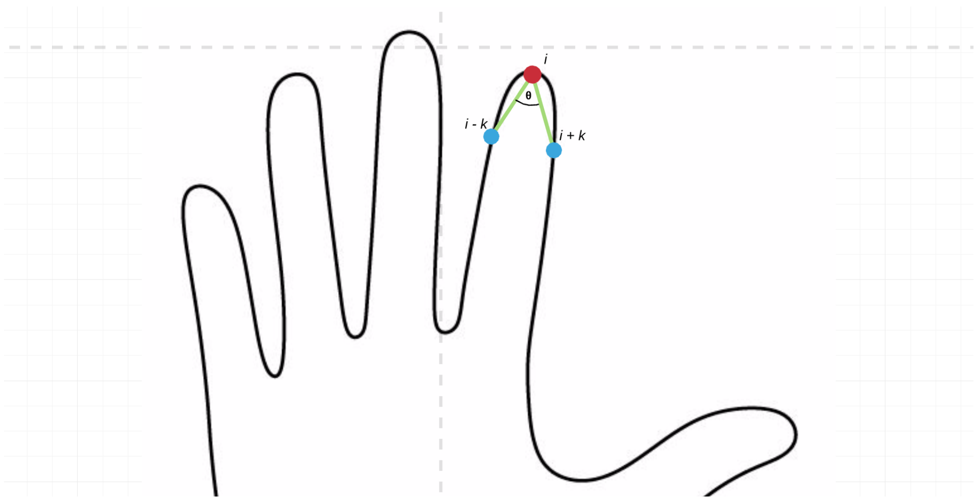
The aim in this step is to label connected components in the thresholded difference image which is returned from background subtraction. Therefore, a search is performed on the binary image to detect all contours in the frame. A contour is a curve of points which all have the same intensity. To detect contours in each frame, OpenCV's findContours() function is used. This function is based on Satoshi Suzuki's border following algorithm. [30] If one or more contours are returned, their size is checked and any contour whose size falls below a threshold is removed. This eliminates any clusters of noise which have been labeled as contours and also helps classify hands from any other, smaller physical items which could have been placed on the system's tabletop surface and segmented in the previous step. If more than one contour remains, the largest contour is determined to represent the hand.

5.2.3 Fingertip Detection

After obtaining the contour of the segmented hand shape, the next objective is to identify the parts of the contour which represent the fingers and fingertips. To do this the k-curvature algorithm is used. The k-curvature algorithm takes each point i along the contour and forms vectors from that point to the points k contour points away in each direction i - k and i + k. [31] k is a constant, representing the vector size. The angle θ between the 2 vectors at the point i is then computed. This is performed for every point in the curve, and the local minima are then computed. The angle between the two vectors is computed using a simple dot product calculation.

$$\theta = \arccos\left(\frac{p1.p2}{|p1||p2|}\right)$$

where *p1* is the vector $[i - k, i]$ and *p2* is the vector $[i, i + k]$.



To obtain the candidate fingertip coordinates, the angle is computed at each point along the hand's contour and compared to a minimum threshold, only keeping points whose angle was less than the threshold. Through trial and error, an angle angle of 60 degrees was found to function well as a threshold for identifying candidate fingertip regions. After filtering out weak candidates using the threshold, a number of candidate fingertip points remain at each convex region. To locate the actual fingertip points from the set of candidates, the local minimum is acquired in each neighbourhood of candidates.

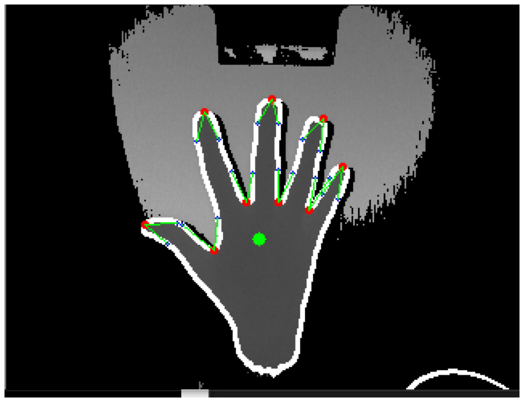


Figure 5.5: Peaks & valleys detected via k-curvature algorith.

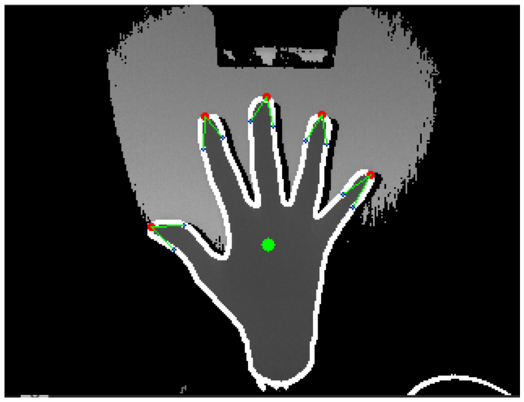


Figure 5.6: The detected fingertip points and palm centroid

The local minima along the contour, which represent the points at which the angle is smallest(ie. the most convex points), can now be categorised as either "peaks"(points at which the curvature is positive) or “valleys”(concave points between fingers). Both peak and valley points can be seen in Fig. 5.5. The valley points need to be discarded as only the peaks are considered to represent fingertips. Hence, a simple check is performed after the local minima are determined. If the y coordinate of point i is less than that of point (i - k), and point (i + k), then it is recognized as a peak. Otherwise, it is a valley and must be thrown away. The final result of fingertip detection can be seen in Fig. 5.6. after the elimination of all valley points.

5.2.4 Hand Centre Determination

Determining the hand centre, after segmenting the hand and finding its contour, is a less involved process than that of fingertip detection, as most of the needed functionality is provided by OpenCV’s *moments* function. Image moments are simple descriptors used to characterize an objects features, in this case a hand contour. The *moments* function can compute the moments of a given contour using Green’s Theorem, which subsequently can provide the contour’s centre of mass. The detected palm centre can be seen in Fig 5.6.

5.2.5 Finger Classification

When more than one finger is detected, a finger is distinguished from the thumb based on a combination of the Euclidean distance from the hand centre, and the angle between the point coordinate and the hand centre.

5.2.6 Gesture Recognition

The system recognizes 4 different gestures, 3 of them static, and 1 dynamic. The static gestures are simply classified based on the number of fingers detected. The dynamic gesture is based on the movement of the detected hand. The gestures are defined as follows:

- “Swipe Mode” On- Two fingers(index finger and extended thumb). Signifies that the user is in the process of swiping a word. All keyboard buttons which the index finger intersects are pressed and their characters added to the character buffer used to accumulate the path of each swipe.

- "Swipe Mode" Off - If one or less fingers are detected, the user can move their hand freely across the virtual keyboard without any keys being selected in order to get to the starting point of their intended swipe path.
- Backspace - The system recognizes a backspace gesture if the user holds out an open palm so that all five of their fingers are detected. A backspace gesture will clear the last-entered word from the display.
- "Next Word" - Is recognized when the user swipes their hand across the screen. This gesture can only be applied when the user raises their hand above the "typing zone", to help distinguish it from swipe gestures. If the user swipes their hand over a sufficiently large distance threshold in the y plane, the previously suggested word is replaced with the next word from the list of ranked candidate words.

The use of the on/off hand gesture during swipe mode can be seen in Figure 5.12 on page 50. The swipe path is initiated and continues while the user's thumb is extended. When the user retracts their thumb, the swipe is finished and the currently entered letters in the character buffer are sent to the string translation thread.

5.2.7 Text Entry

Once the gesture to switch to begin " swipe mode " has been recognized, any key that the index fingertip's x,y coordinates intersect is activated so that its character is added to a character buffer. The character buffer contains all the keyboard characters activated in the swipe so far, in the order in which they were added. When the user closes their thumb to form the " swipe mode " off gesture is recognized, signifying that they are finished swiping a word, the string of characters in the character buffer is sent to the string translation function, and the buffer is then cleared for the next word.

5.3 String Translation

In order to resolve a user's swipe gestures into their desired words, the functionality of word gesture keyboards must be modeled. While there are a number of competing mobile apps which provide different WGKs, these all seem to be patented, closed solutions. There was no open source library available to me to provide swipe functionality to the keyboard system, so I was tasked with developing my own implementation. While the inner workings of the most popular swipe keyboard apps are not publicly available, several tech blogs and articles exist which postulate about how the problem could be solved at a high level, which I consulted to develop a strategy for the solution to this problem. [32]

The main challenge of implementing swipe-based text entry was to accept the input string of keyboard characters which the user's finger has swiped over and determine the word that they intended to type. This type of problem falls into the fuzzy string searching problem domain. When the user wishes to enter a word by swiping over its letters in a singular motion, they will have to swipe over all the characters that occur between their desired letters in a QWERTY keyboard layout. So to swipe the word "basket", for example, the letters "BVCXZASDFGHJKUYTRERT", or a similar pattern, are likely to be input. The goal was to take an input pattern of this form and perform a dictionary search to ascertain the most likely words that the user intended to enter.

During the initialization of the program, a large dictionary file containing almost 200,000 English-language known words is loaded into a dictionary. The system needed to be able to search this dictionary to find the word which is the closest match to the received input string and display it, along with the next n best candidates. This means that if the first suggested word doesn't match the word that the user had intended to swipe, provided that their swipe path was reasonably accurate, then they should be able to select their word from the small set of alternative word suggestions. Any delay in determining the best match would have a detrimental effect upon user experience, so speed was imperative here. Furthermore, for the system to be as user-friendly as possible, the suggestions needed to be sufficiently accurate so that the user's intended word was always presented, with some allowances for missing or extra letters in the input string, eliminating the need for the user to ever have to re-enter a word. Hence, speed and accuracy were two fundamental requirements during this stage of the system's implementation.

5.3.1 Levenshtein Distance

In order to find the dictionary word that was the closest match to the set of swiped letters, the Levenshtein distance algorithm(sometimes referred to as edit distance) was used. Levenshtein distance[33] is a metric used to measure the similarity between two strings. When provided with a source string and a target string, the algorithm produces a distance value, where distance is defined as the minimum number of insertions, deletions, or substitutions required to change the source into the target. Levenshtein distance has many applications, such as in spell checking programs, plagiarism detection and speech recognition. The time complexity of the Levenshtein distance algorithm is $O(n^2)$.

To show how Levenshtein Distance works, take the example of a situation in which we wish to find the number of changes required to convert the word *kitten* into the word *sitting*:

kitten → **s**itten	(substitution)
sitt**e**n → sitt**i**n	(substitution)
sittin → sittin**g**	(insertion)

With 2 substitutions and 1 insertion, the source word is modified into the target word, giving a Levenshtein distance of 3.

Levenshtein Distance Algorithm

The general form of the Levenshtein distance algorithm is as follows [34]:

1. **Assign the length of the source string *s* to the variable *n*, and the length of the target string *t* to be *m*.**
   If $n == 0$, return m.
   If $m == 0$, return n.

2. **Form a matrix *d*, of dimensions *m* x *n*.**
   The first row is initialized as $0..n$.
   The first column is initialized as $0..m$.

3. **The characters of each word are compared.**

```
for **i** from 1 to $n$:
  for **j** from 1 to $m$:
    if $s[\mathbf{i}] == t[\mathbf{j}]$:   cost = 0
    else:   cost = 1
```

4. **The value of the matrix d at index [i, j] is set to the minimum of:**
   The value of the cell directly above **d**[i,j], plus 1. **d**[i-1,j] + 1.
   The value of the cell to the immediate left of **d**[i,j], plus 1. **d**[i,j-1] + 1.
   The value of the top left diagonal cell adjacent to **d**[i,j], plus the cost value. **d**[i-1,j-1] + cost.

5. Once the nested *for* loop has finished iterating over the characters of the two words, the final Levenshtein distance score will be stored in the cell [n][m] of the resultant matrix.

5.3.2 Search Space Reduction

To calculate the Levenshtein Distance between the swiped input string and every word in the dictionary and identifying the word that gives the smallest distance as the strongest candidate would be needlessly inefficent. This approach would be far too costly for a real-time application due to the size of the dictionary. There are a number of improvements which can be applied to the dictionary search to make the lookup process less computationally intensive. A number of different metrics were examined based around identifying characters in the input string which were likely to be intentional and could be used to inform the search. These intentional characters, which belong to the user's intended word, are referred to as critical keys.

Key Presence

Firstly, the system operates under the assumption that the first and last letters of the input string will correspond to the first and last letters of the intended word. If these letters do not correspond to those of the intended word, it is assumed that there was an error on the user's part. A nested dictionary was implemented to carry out a fast search for words based on their first and last letter. For example: wordList['a']['t'] would return the following set: ['acrobat', 'act', 'activist', 'adjacent', 'ant', 'apparent','art'].

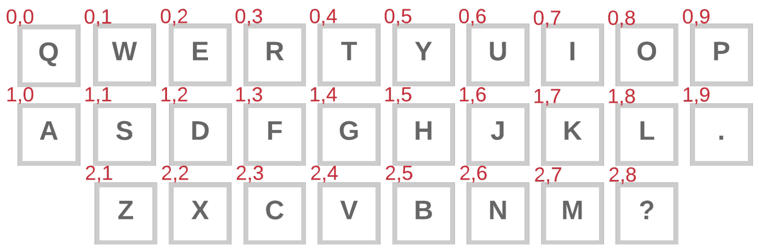
This constraint was needed in order to greatly reduce the search space to the set of words that share the first and final character of the input string. The speed of the " input string to dictionary word " translation process is hence improved, allowing it to run smoothly in real-time. Having to compare and rank too large a number of candidate dictionary words would result in a noticeable lag in the output display of the resolved word. However, enforcing this constraint, which requires the user to correctly swipe over the first and last letter of their intended word, will incur a higher error rate than a solution which can allow for minor misspellings of the input word, since errors of this nature are relatively common in typing applications. Ultimately, there is a trade-off between the speed of word resolution, and the allowance of errors.

The size of the set of candidate words can be further decreased by eliminating all words whose letters are not contained in the swiped string. A regular expression composed of the ordered letters of each candidate is matched against the input swiped word to further decrease the number of candidate words. This is necessary in order to improve search time and reduce the set of possible candidates, however it does add the constraint that the user swiped over all of the characters of their desired word in the correct order. If the user accidentally omits a letter, their word will not be resolved. For example if the swiped input word is "BVCXZASDFGHJKUYTRERT" and the set of candidates is ["BAT", "BASKET", "BET", "BLACKOUT", "BOOT", "BRACELET"], each word is converted to a regular expression which matches any string of characters which contains the same set of letters, with the same order, with any number of alphabet characters in between: [ˆB.*A.*T$, ˆB.*A.*S.*K.*E.*T$, ˆB.*E.*T$, ˆB.*L.*A.*C.*K.*O.*U.*T$, ˆB.*O.*O.*T$, ˆB.*R.*A.*C.*L.*E.*T$]. After the swiped string is matched against each regular expression, the set is reduced to ["BAT", "BASKET", "BET"].

Inflection Points

For each word in the reduced set of candidates, the Levenshtein distance between it and the swiped string is calculated. The words are then sorted, with the lowest-scored word representing the closest match. The best 5 candidates are kept and shown to the user, who can select from them if the first ranked word does not match their intended entry. Using this approach, the results were found to be fairly accurate, with the intended word being returned in the 5 best candidates approximately 75% of the time. However, one particular flaw of this approach was that words of smaller length were often not correctly resolved. This is because, due to the input search string being significantly longer than the intended word(typically 2-3 times its size), longer words tended to have a closer Levenshtein score to the input swipe. In order to get around this issue, a second distance-based check was added, this time based on analysing changes of direction in the users swipe path.

The ranking process can be improved by inferring that when a significant change in direction takes place, the letter around which the turn occurred is a critical key. To calculate direction changes in a swiped string, the keys of the QWERTY layout are each assigned a Cartesian coordinate pair. Each key's coordinate pair is determined relative to the first key, Q, whose coordinate is the origin point (0,0):



For each character in the swiped string, the transition to the current key (y2, x2) from the previous character (y1, x1) is positive in the X direction if x2 >x1. Likewise, the transition is positive in the Y direction if y2 >y1. If, for any pair of consecutive keys in the input swipe string, the direction between the two keys differs for one, or both, of X and Y compared to the previous direction, a critical key is flagged.
For example, if the user aims to enter the word "HAND", their swiped input string may be of the form "HGFDSASDFGHNHGFD":

- The difference between the coordinates for the first letter "H" and the following letter "G" is determined: (1,4) - (1,5) = (0,-1), which gives a negative value for the X direction.
- At each of the next four key transitions G → F, F → D, D → S, S → A, the direction remains negative in the X lane along the Y axis.

- When the swiped string's characters transition from A $\rightarrow$ S, the direction becomes position (1,1) -(1,0) = (0,1). Hence, A is a point of inflection at which the direction of the swipe path has been reversed from -1 to +1.
- Another such direction change occurs for the transitions H $\rightarrow$ N, N $\rightarrow$ H. The direction along the Y axis goes from positive((2,6) -(1,5) = (1,1)) to negative ((1,5) - (2,6) = (-1,-1)), indicating that N is also a point of inflection.

Using the Levenshtein distance to match this string "HGFDSASDFGHNHGFD" to 'hand' gives a score of 12. However, the critical keys 'a' and 'n', which are highlighted by the reversals of direction at the sequences 'sas' and 'hnh' in the swipe path, can be used to form a new search string. This search string begins and ends with the first and last characters, since these are always regarded to be important. Hence, by combining the first and last characters "H" and "D" with the critical keys "A" and "N", in order, we are left with the search string "HAND", which is obviously already a distance of 0 away from the target string, giving us a perfect match.

Normalized Levenshtein Distance

Using points of inflection as critical keys for Levenshtein distance computation did not outperform the initial approach(which used key presence). However, combining the Levenshtein distance scores from the two approaches proved to cover the vast majority of cases, significantly outperforming the original method which only took letter presence into account. In order to combine two Levenshtein distance values, they must be normalized, before being summed and divided by 2 to get the mean. To normalize a Levenshtein distance value, the following formula is used:

$$normalizedLD = 1 - \frac{levenshteinDist(word1, word2)}{max(len(word1), len(word2))}$$

In summary, when a user inputs a swiped string, the Levenshtein distance is computed between that string and each dictionary word whose letters are all contained, in order, within the input string, provided that their first and final letters also match. A second pass is made in which a string is formed from points of inflection plus the first/final letters. The Levenshtein distance is then calculated for each candidate word between that word and the string of critical keys. For each word, there are now 2 Levenshtein distance values. These values are normalized and averaged. Using this new average value, the candidates are then re-ordered and the best 5 are returned to the user. Due to the computational complexity of the swipe resolution algorithm, it is run in a separate thread to the main thread in which image processing is performed. When a user has finished swiping a word, the thread on which the string translation function runs is signalled, and the contents of the character buffer are sent to a shared variable which both threads can access through use of mutex locks. When the swipe resolution process is complete, the set of candidate words is returned via another shared variable.

Other Metrics Considered

Another metric which I investigated involved analyzing the timing of the user's input swipe. Based on the assumption that the user will swipe more quickly over the intermediary keys and slow down to approach the critical keys on the keyboard, the timing approach assigns higher probability scores to letters over which the user spent longer intervals. However, through testing I found that this approach was not useful as the basic assumption that the amount of time spent over a letter implies its correctness isn't always true. Furthermore, when the user is typing at very high speeds, the limited framerate of the camera may be insufficient for measuring significant differences in intervals between critical and non-critical keys. Hence, this metric was not used in the final system.

Similarly, one approach that appears to be used by popular mobile-based word gesture keyboards is to use the frequency of a word's occurrence in previous text written by the user as an indicator of likelihood that the user intended to swipe it in the current instance. This approach is more suited for use in the keyboard systems of personal devices(laptops, tablets, phones, etc). In a public terminal such as SureWash, or any other hospital-based kiosk using a touchless keyboard, a profile of word usage cannot be built up for a single user since the system is shared by many users, so for this system, word frequency was not used.

5.4 Display

In addition to the backend logic required for capturing the user's interaction and resolve their actions to controls, a vital aspect of the overall system is its frontend GUI. A visualization of a virtual keyboard needs to be displayed to provide the user with something to interact with. For each captured frame, once the position, depth, and gesture of the hand has been computed, and the button boundaries with which the hand intersects have been determined, the output GUI needs to be updated so that the users actions are re-displayed in real time.

As discussed previously, for the implementation of this stage in the system's pipeline, rather than constructing a virtual interface in which a graphical representation of the users hand/fingers is shown, an augmented reality approach was taken in order to create an intuitive GUI which simulates a real-world keyboard as closely as possible while providing additional augmented features aimed to enhance usability. Since the camera is positioned to point down towards the tabletop surface from above, the UI was required to provide a top-down visualization of a keyboard which is rendered onto the image of the tabletop surface. Hence, the GUI is constructed by displaying each colour frame captured by the camera with a blended graphical keyboard which is positioned onto the tabletop surface. The keyboard interface is drawn in such a way that the users hands occlude it to create the illusion that they float above it. This is done by drawing the interface around any hands detected in the scene.

5.4.1 Virtual Keyboard UI

The rendering of the basic, un-masked virtual keyboard user interface is a relatively straightforward process. The goal was to create a virtual interface which closely resembles a physical keyboard. At present, there are countless examples of such virtual keyboard interfaces which can exist in popular devices and applications. For example, any smartphone has a virtual keyboard interface in its SMS and e-mail applications, games consoles will often have an on-screen virtual keyboard interface for text-entry tasks. Across a myriad of different platforms, these virtual keyboard displays do not typically diverge from the same class keyboard structure: consisting of a single button for each key, a QWERTY-based key layout, and an input field which displays each received letter as it's typed. The only real variable aspects to such keyboards tend to be preference-based UI features, such as colour, font, key-size and key spacing.

The virtual keyboard interface for my system was implemented to follow this recurrent design. The keyboard UI is designed to be minimalistic, composed of a simple set of buttons without any exterior frame, or background. This intends to enhance its blending with the existing scene. The keyboard is rendered into the scene by drawing it over each colour frame in real-time. Rather than use a separate graphics graphics framework such as OpenGl or Unity3D to perform the UI rendering, I opted to utilize OpenCV's drawing functions, which provide all of the functionality required for my design. OpenCV's drawing functions allow anti-aliased shapes and fonts to be rendered efficiently over an image frame. This set of functions is mainly intended for the drawing of 2D shapes such as circles, rectangles and lines. Since my system has a top-down view of the surface, the virtual keyboard is also drawn as if it's being viewed from above. Hence, two-dimensional shapes suffice to draw buttons from this angle. The OpenCV *rectangle* drawing function was used to draw each key/button as a rectangle which is masked with the key's alphanumeric character based on the QWERTY layout. The OpenCV-rendered grid of virtual buttons, overlayed into the real-time scene can be seen in Fig. 5.7 below. The buttons were made opaque by lowering their alpha value so that they blend more naturally with the scene and that the tabletop surface below the interface can still be viewed.

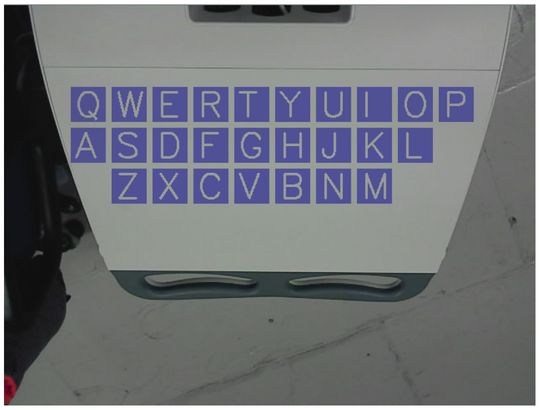


Figure 5.7: The minimalistic keyboard button grid interface

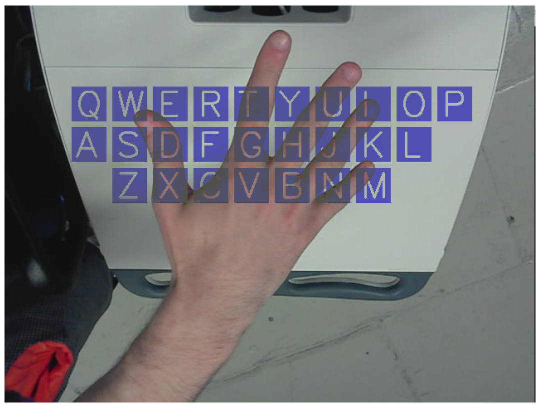


Figure 5.8: Before masking is applied, the interface is drawn over the hand

5.4.2 Hand Masking

To implement an “under the hand” style keyboard interface in which the user’s hand occludes the virtual keyboard, a mask was created for the segmented hand and subtracted from the virtual keyboard object so that when the keyboard UI is overlayed onto each camera frame, the button elements are not drawn over the hand at any point. Rather, they appear to lie beneath the hand.

During the image processing stage, it will have already been determined whether or not a hand is present in the scene. If no hands are detected, the unaltered virtual keyboard is overlayed onto each camera frame. Otherwise, a binary mask of the segmented hand needs to be computed. A mask of the hand will have already been constructed following the background subtraction step of the image processing stage. However, this operation was performed on the depth stream, which provides single channel frames at 320x240 resolution, while the GUI re-displays colour frames which have a resolution of 640x480. Due to this difference in resolutions, a mask of the hand based on depth cannot be simply applied to the colour image of the same frame.

A mapping is needed to match each pixel in the depth image to a corresponding pixel of the higher resolution colour image. One issue which arises when trying to map between the frames from the two different cameras is the displacement between the physical cameras. Normally, a camera calibration would need to be performed in order to establish a perfect mapping between the two. However, the DepthSense SDK provides a UV Mapping function which accounts for this displacement and can determine pixel correspondences between the two frame resolutions.

The colour frame could be down-scaled from 640x480 to 320x240 to simplify the mapping, but this would result in a loss of quality in the output display. The alternative is to up-scale the depth frame to the resolution of the colour frame. This can be done with the UV Mapping function. However, since the resolution of the colour sensor is twice that of the depth sensor, depth pixel data is only available for a subset of all colour pixels, so an interpolation scheme must be applied in order to fill in the gaps. After performing the warping from depth image to colour with various interpolation methods tested, I was unable to obtain a mask of the hand at the colour resolution with the same accuracy of the mask acquired from the depth stream.

Rather than continue in the attempt to warp the depth mask to a higher resolution while trying to preserve the hand shape's edge accuracy, my solution was to use this rough mask of the hand in the colour stream to seed an adaptive histogram of the hand's skin colour. This histogram can then be applied to the original colour frame to get an accurate, high-resolution hand mask.

Depth-Seeded Adaptive Histogram

The basic idea of the approach is to the hand mask obtained in the depth stream and UV map it up to the colour stream so that it now functions as a rough mask of the hand. This rough mask is eroded, to remove surrounding non-skin pixels, and then used to seed a HS skin-colour histogram. Hence, the solution is an adaptive approach similar to the online skin color modeling methods documented in [35] and [36].

While these approaches use an initialization stage to build the skin colour model based on a detected face, or by having the user hold out their hand for a short period, my approach seeds a histogram based on the roughly segmented hand region which has been warped from the depth resolution to the colour resolution. This has the advantage of not requiring the initialization stage, as it makes the assumption that any sufficiently large contour held out above the table surface will be a hand from which it can obtain the skin colour model. It requires that the user's sleeves are rolled up in order for the model to be accurate. This adaptive skin colour modelling approach also has advantages over traditional skin detection approaches which rely on offline models, as skin colour can have large variation between different individuals, and the same person's skin may look vastly different under different lighting conditions. With an adaptive approach, the skin model will always be built online and hence will be solely based on the current users skin colour and the illumination of their hands.

The colour model in question is 2-channel histogram consisting of the hue and saturation components of the hand region colour. The rough hand area is converted from RGB to HSV colour space and OpenCV's calcHist function is used to obtain a histogram of the hand region. HSV colour space was chosen because, despite RGB's advantages as a colour space for image display, it is not optimal for skin-based segmentation due to its susceptibility to being affected by changes in lighting. This susceptibility arises because all 3 channels in RGB are highly correlated, each containing a luminance value in addition to chrominance. HSV, on the other hand, separates the hue(colour), saturation(colour purity) and brightness value from each other. HSV has been shown to outperform other RGB and other colour spaces for skin colour detection applications. [37] Hence, by ignoring the 3rd channel and dealing solely with hue and saturation, we have a colour model which is invariant to changes in illumination and the occurrence of shadows.

Once the histogram has been calculated, a simple back projection can be applied to the colour image in order to create a thresholded image of all skin colour regions detected. Back projection is a technique which allows a feature histogram to be applied to an image so that every pixel of the image which fits the model distribution can be thresholded. OpenCV's calcBackProject() method provides the necessary functionality to complete this step. At this point, the result of back projection should provide an accurate mask of the hand, as can be seen in Fig 5.9.

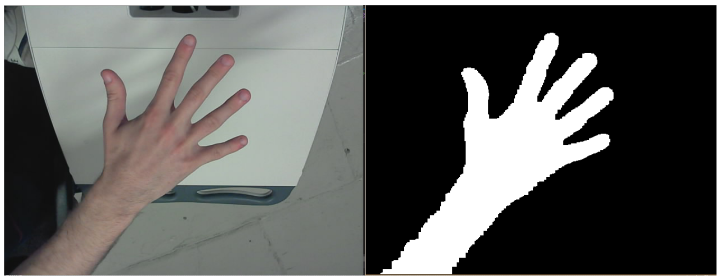


Figure 5.9: High-res hand mask computed from the depth-seeded histogram

The hand outline can be removed from the button grid by performing a bitwise_xor operation with the two binary mask of the hand and the binary mask of the button grid. The result is shown in Figures 5.10 and 5.11 below. Fig 5.10 shows what the UI looks like before masking is applied. The keyboard interface is simply drawn over the frame, not interacting in any way with the hand. In Fig 5.11, the hand mask has been XORed with the button grid so that the buttons are drawn around the hand, creating the appearance of it resting at a higher plane of the depth than the keyboard. Also, the implementation of target expansion, discussed in Chapter 4., can be seen here. The key over which the index fingertip rests is expanded in size and its colour has been changed to highlight it.

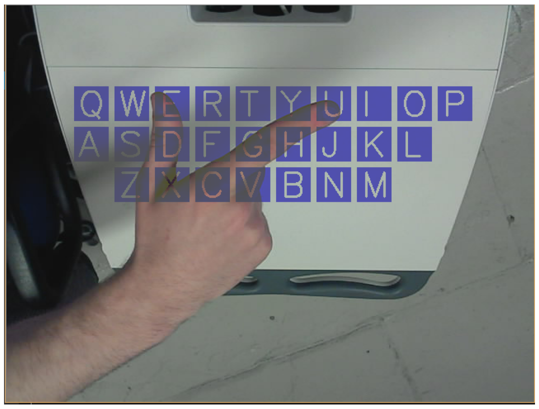


Figure 5.10: Before applying the hand mask, the hand appears to be underneath the keyboard.

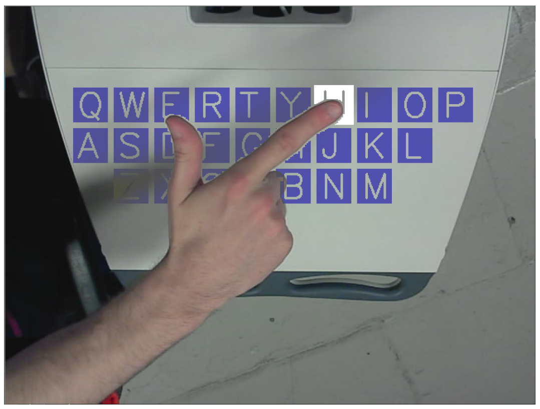
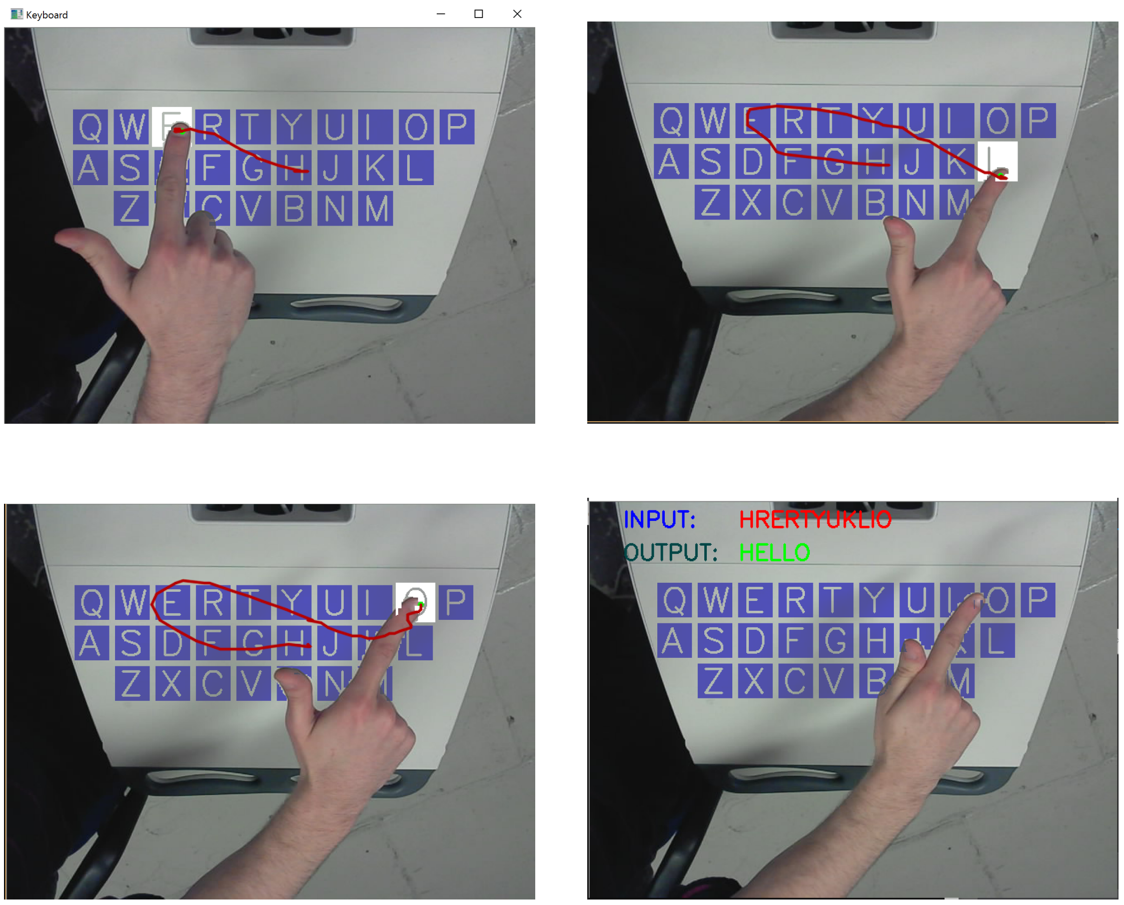


Figure 5.11: Following the masking process, the hand floats above the keyboard.

The index fingertip point had already been detected in the depth image, but as can be seen in Fig 5.11, it has now been warped to the colour image. This is done through the DepthSense SDK's UV mapping function which warps a frame from the depth stream to a frame of the colour stream and vice versa. This was not suitable to use for warping up the hand because the warping process caused the hand shape to become jagged and misaligned. However, this same function performs well for the up-mapping of a single point, which is the case when warping the fingertip. Hence, we now have the x,y coordinates of the index fingertip point at the higher resolution, so that the buttons with which it intersects can be detected.

With the fingertip detected, the path of the finger in a single swipe can be drawn to assist the visualization of a user's gesture and provide feedback to inform them of whether their swipe has crossed the desired letters. The swipe path is drawn using OpenCV's *polylines()* function. This function takes a list of 2D points and draws a number of polygonal line segments between them. Each detected fingertip point during a single swipe is stored in a global array which can be then used in the *polylines()* function to draw the swipe path. In order to remove minor jitters from the output curve, a fingertip point is only added to the polyline input set if the finger has moved over a minimum Euclidean distance threshold since its last recorded point. This prevents the swipe path from containing multiple minor direction changes which often occur due to the natural tremors of the hand when extended in mid-air.



```
INPUT:     HRERTYUKLIO
OUTPUT:    HELLO
```

Figure 5.12: A user enters the word “hello” with a single, continuous swipe gesture.

5.5 Eliminating Bottlenecks

One recurring issue encountered throughout the development process was that of computational bottlenecks arising in the system. A bottleneck refers to a single point of congestion in an application which negatively affects the performance of the application as a whole. There are many ways in which bottlenecks can arise, especially in vision systems, which often need to perform expensive per-pixel image searches on each processed frame.

In a real-time human-controlled interface system it is imperative that the system is interactive, and instantaneously responsive to the user's movements. Any bottlenecks in the code act as obstacles to this. Since a bottleneck will always limit the overall throughput of the system, they usually manifest in delay between a user's real-world movements, and the corresponding update of their actions to the system's display. Lag of this nature has a highly detrimental effect upon usability. As a result of this, I tried to emphasize efficiency within code, seeking optimization wherever possible.

As outlined previously, there are 3 main conceptual aspects to my system: the image processing stage, text entry determination, and visual display. In each of these areas, complex operations are being performed for each captured frame, which often gave rise to various bottlenecks. When bottlenecks arose, I used simple timers and code profilers to identify whereabouts in the source code performance was being stalled. Some of the steps which I took to reducing bottlenecks in each pipeline stage include:

- Performing the majority of my image processing operations on the lower resolution depth image than the colour image reduced the time required for operations such as findContours(), and was also beneficial due to depth data providing more reliable data than colour(which was affected by illumination changes) in my application.

- Using multithreading to perform independent functions in separate threads. In the case of my program, all string-to-word operations are performed in one thread, while image processing operations are performed in another. The two threads communicate through use of signals and shared variables with locks.

- Rendering as much of the graphic interface as possible outside the main draw loop. Rendering and blending the keyboard interface in the initialization stages greatly reduced lag in my display.

- Ensuring that system sounds are played asynchronously.
- Using an efficient dictionary search in the string-to-word function which eliminates non viable candidates by quickly retrieving all words that contain the same first and last characters as the swiped input string.

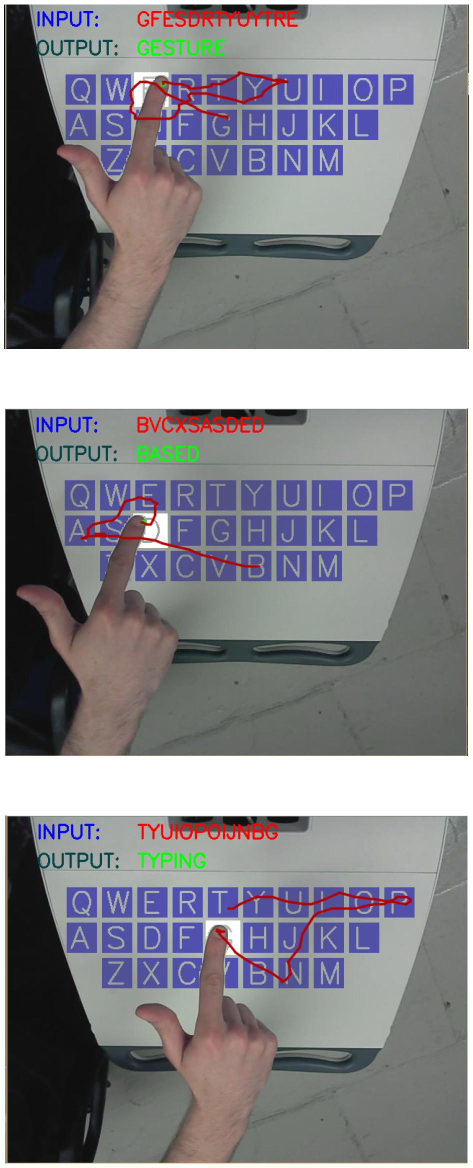


Figure 5.13: Some text-entry examples

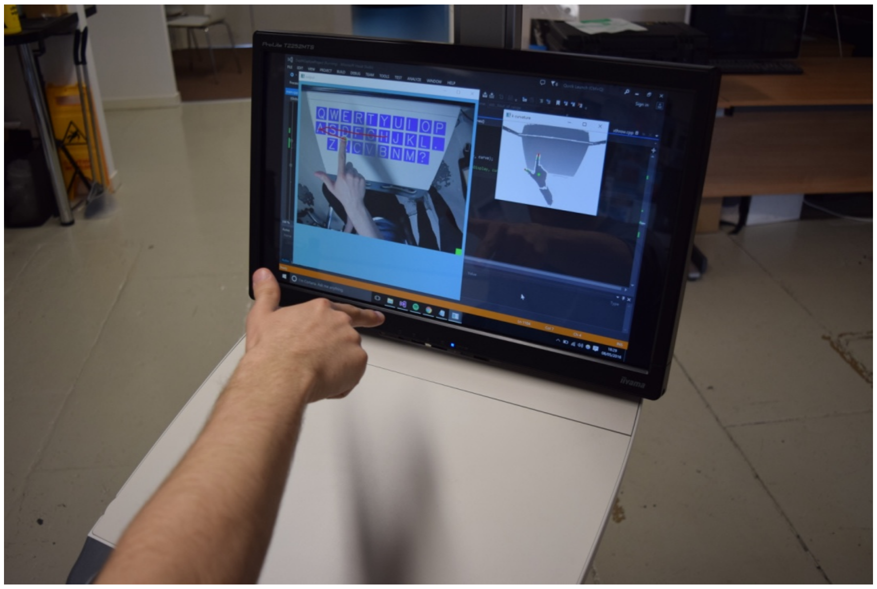


Figure 5.14: The user relies solely on a visual interface.

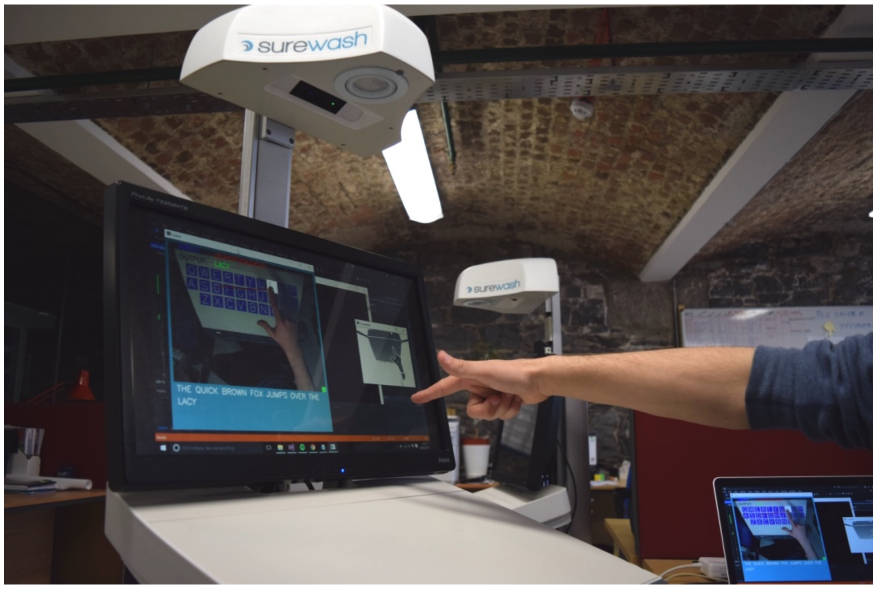


Figure 5.15: A user extends their index finger and thumb to initiate a swipe gesture.

Chapter 6

Results & Evaluation

6.1 Initial Pilot Study

Once an initial working prototype of the system was developed, I conducted a small pilot study in order to gain an initial insight into how users responded to the system and also to learn which aspects of the system required improvement and/or design changes. For the pilot study I asked 3 test subjects to use the system and enter a small set of English-language sentences using the touch-less interface. Their usage was observed and all errors which arose were recorded. Following the test, subjects were asked about how they felt about various different aspects of the system and their response was recorded. The study indicated several elements of the system which required improvement, the main issue being finger detection failures which were arising due to users' hands tending to naturally drift below the minimum depth plane. In response to this feedback, appropriate changes were made to the system in order to improve the hand segmentation implementation and allow for a lower depth minimum depth plane. This allows hands to fall closer to the tabletop surface without detection failing.

6.2 User Study

A user evaluation test was designed with the purpose of providing data results for the text-entry performance of the system and indicating the suitability of a touch-less, swiped-based AR keyboard for text entry tasks.

Participants

A group of 7 test subjects took part in the study. The participants were all either work colleagues or students, between the ages of 22 and 45, none of whom had used the system before. Of the subjects tested, 6 were male and 1 was female. In terms of hand preference, 6 were right-handed, and 1 partipant was left-handed. Prior to the testing, the subjects were asked about their experience with word gesture keyboards(WGKs) on mobile platforms. Since WGKs themselves have a learning curve, I was interested in identifying whether there was a correlation between familiarity with the mobile implementation of swipe-based typing and performance with my keyboard-based implementation. Of the 7 participants, 3 had never used a WGK, 3 had used a mobile-based WGK before, but did not use it regularly, and 1 participant regularly used a WGK on their mobile phone as their preferred mode of typing. All participants were fluent in written English.

Study

The system was evaluated by having participants complete some basic text entry tasks using the keyboard. The application was run from a laptop connected to a SureWash unit, so that the screen and camera of the unit provided the user interface. To familiarize users with the system, they were each given an introductory explanation of how the system works, how to use the system, and the tasks that they would be required to perform. They were then given 1-2 minutes to type freely on the system in order to familiarize themselves with its usage.

Once each user felt comfortable with the basic mechanics of the system and had practiced entering some arbitrary words using swipe gestures, the test was begun. In order to complete the test, users had to enter 3 English-language sentences(picked from a sample set of standard sentences for text-assessment) while being timed. Each user was shown the same set of sentences. User performance was recorded in a logging function written specifically for these tests. Each sentence was displayed on-screen, with the due word highlighted. When the user swiped over the letters of the highlighted word, the logging function recorded their entered word, whether it was correct or incorrect, and the time taken to swipe the word. If a word was swiped correctly, their entry was accepted and the next word was highlighted. If a user failed to swipe a word, they were requested to try again.

Users were also given a questionnaire that assess their experience with the system using a 5 point Likert scale which questioned the user's feelings regarding the system's: ease of use, enjoyability, learnability and responsiveness.

Text entry speed was measured using the Words Per Minute(WPM) metric, which is the most commonly used empirical measure for assessing text entry performance. [38]. WPM is defined as:

$$WPM = \frac{|T| - 1}{S} \times 60 \times \frac{1}{5}$$

where:
T is the transcribed string,
$|T|$ is the transcribed string's length.
and S is the number of seconds, starting with the first keypress, taken for the user to produce the string T.

The subtraction of 1 from the total string length in the above formula is intended to account for the entry of the first character, because logging is not initiated until this character is entered. Since the value $|T|$ represents the transcribed string's total length, it factors all the words entered in addition to spaces and punctuation into its calculation. In my system, spaces are added automatically to the end of each word unless other punctuation is specifically entered by the user. Although a single swiped input word will contain many extraneous characters, only resolved words from the dictionary list were added to T, rather than the raw input swipe string. Pauses between sentences were not factored into the measurement.

The typical WPM metric measures the time taken to enter all text, regardless of errors in the input. It can be seen as a Gross WPM score. Net Words Per Minute is a more useful metric, since it takes errors into account and provides a more accurate measure of keyboard performance. Net WPM applies the same formula as above but computes the length of T by only summing the characters from correct words, plus spaces and punctuation. Hence, it will typically be lower than the gross score, which counts incorrectly swiped words. For example, if the user swipes the word “HAND” but their swipe action unintentionally appends an unwanted neighbouring letter such as “S” to the end of the word, the testing program is expecting “HAND” but receives “HANDS” so records this as an error. Errors of this nature, where the swiped entry is close to the desired word, but not exact, are factored into the Gross WPM score but omitted from the Net WPM score.

While the majority of the participants could not be retained for further testing after the initial user study was carried out, one participant was given 2 additional tests after their first-time test. Each session followed the exact same protocol, the user spent an initial 1-2 minutes of free, un-recorded typing to re-familiarize themselves with the system. They were then required to enter the same 3 sentences from the first test. All 3 tests were conducted 3-5 days apart from one another, with the intention of studying the effects of the system’s learnability - how its usability improves over time.

6.3 Results

Figure 6.1 shows the mean WPM and Net WPM recorded for all first-time users. The mean Gross WPM score for all participants was calculated to be 7.57, with a standard deviation of 0.79. The mean Net/Correct WPM score for all participants was 6.26 with a standard deviation of 0.48. The mean error rate(calculated as the difference between the gross and net scores) was 17.3%.

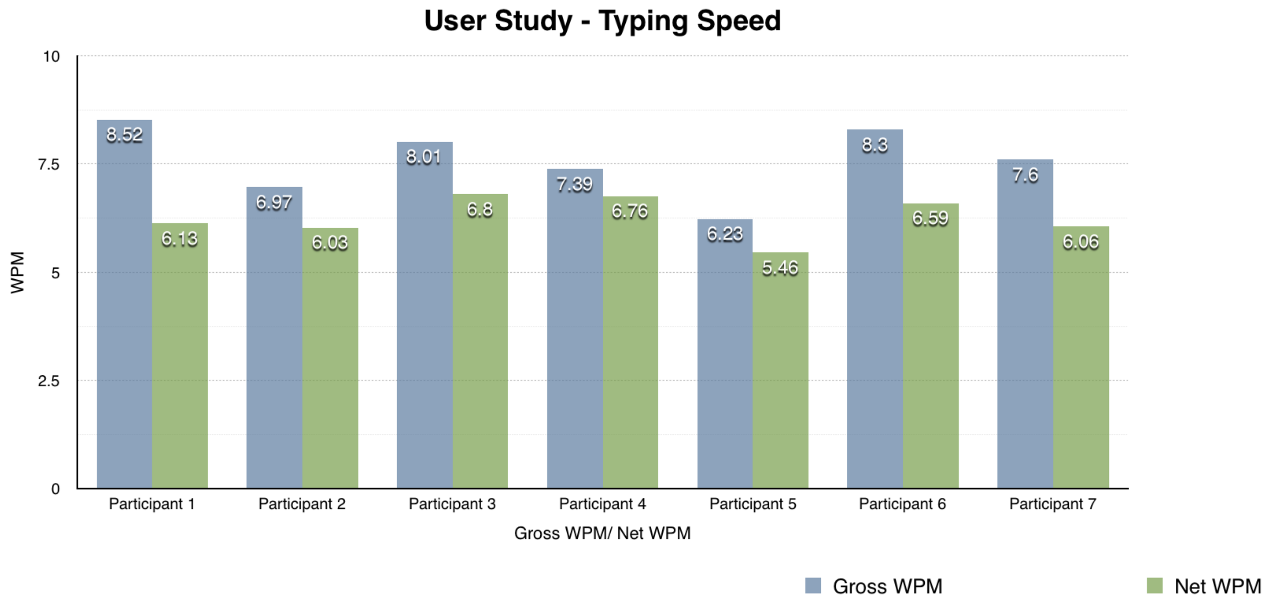


Figure 6.1: A comparison of Gross WPM and Net WPM recorded for each first-time participant

Users were provided with a set of questions regarding the experience with the system and answered these questions using a 5-point Likert scale. Their responses are collected in the chart at Figure 6.2.

Figure 6.3 shows evidence for the system's learnability, as a single user's score improved from 7.39/6.76 WPM to 8.9/8.2 WPM over 3 sessions.

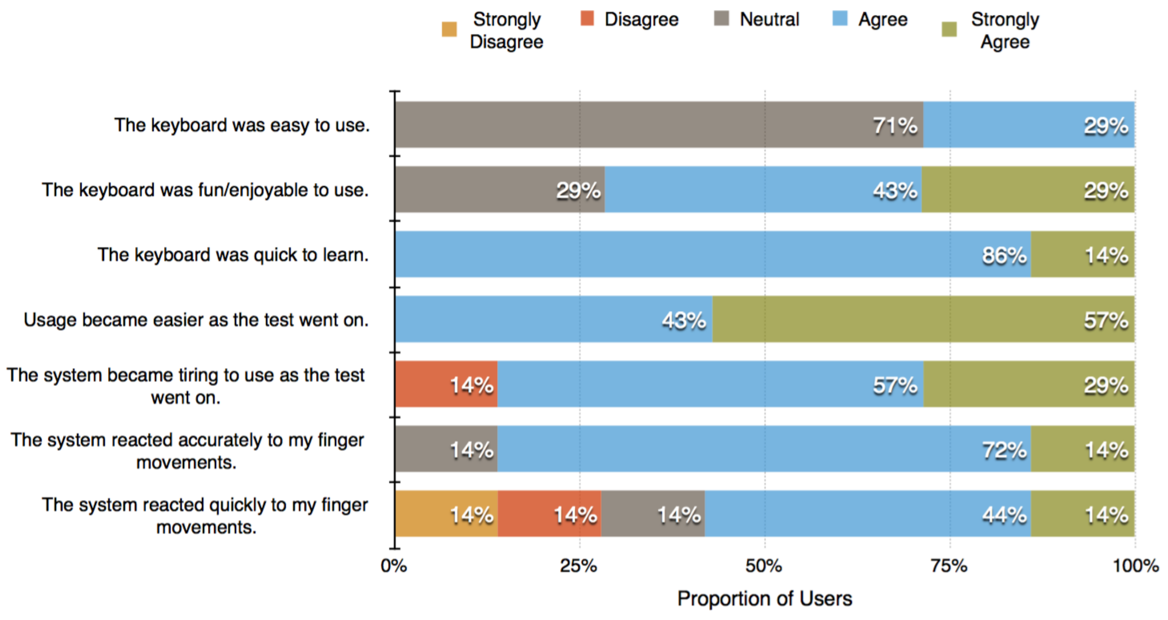


Figure 6.2: Respones to the 5-point Likert questionnaire given to users folowing testing.

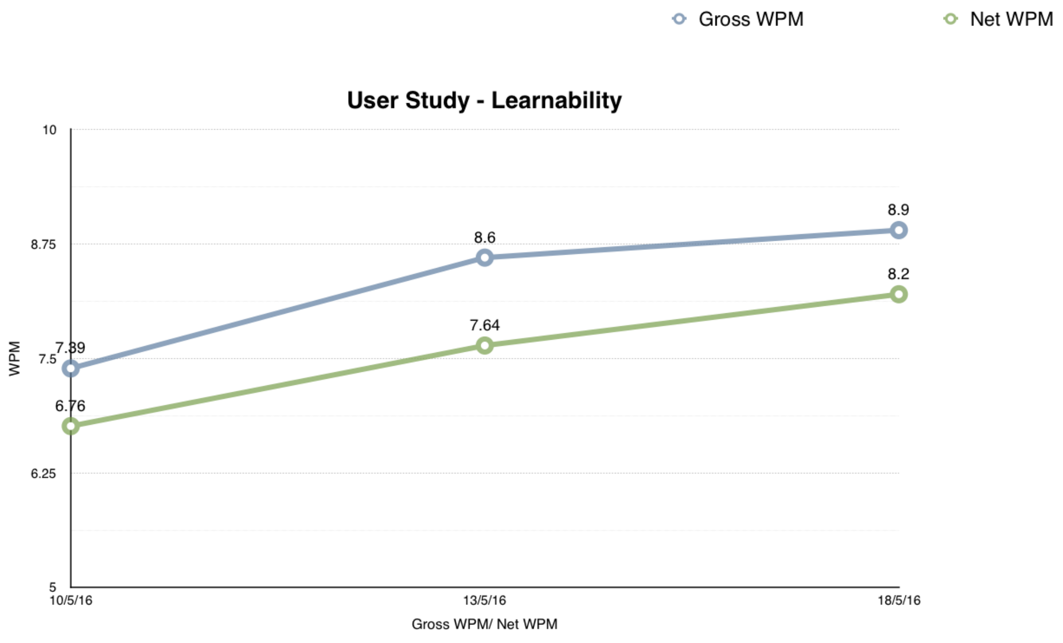


Figure 6.3: The line chart shows the improvement in a single user's performance over three sessions.

6.4 Evaluation

While the keyboard application provides a usable, touchless text-entry system, it performs poorly in terms of text-entry speed and error rate. The user study highlighted that barriers to speed and accuracy stem from the following issues:

- Firstly, one of the fundamental issues experienced by users was coordinating the free movement of their hand in 3D space to control the on-screen system. Hand eye coordination without any physical interface or haptic feedback makes high levels of speed and accuracy difficult to obtain, especially for first-time users. This was generally less of an issue when users took their time and typed more slowly, but prevents the system from being able to achieve typing speeds which are comparable to the benchmarks set by existing physical and touch-screen keyboards. The need to coordinate the hand accurately in relation to the output visualization slows operation of the system in the sense that in order to achieve better accuracy, users tend to need to slow their swipe gesture. The amount of keys, their individual sizes and their relative proximity to one another makes accurately selecting between neighbouring keys a difficult task for any user to perform quickly.

- The main cause of typing errors during the testing process was the swipe translation approach employed in the system. My fuzz string matching function requires always that the first and final character in each swipe are correct. However, due to the control and coordination issues outlined above, in addition to the minor tremors which naturally tend to occur in a user's hand, unintended key presses can occur quite easily. When accidental key presses of this nature transpire, especially at the beginning or end of an entered word, the swipe resolution algorithm is unable to determine that a new letter has been erroneously added to the swipe and hence builds its search around the accidental letter. A superior string resolution approach which is able to find candidate words with more allowances for errors could greatly alleviate this issue.

- Another, less frequent, cause of errors during the testing process was when detection of the finger failed. When the system detects one or fewer fingers during a swipe gesture, it is concluded that the swipe has been finished by the user. Therefore the swipe path is closed off and the current set of letters in the character buffer are sent to the swipe translation thread for string resolution.

Certain hand positions can cause occlusions of the fingers for multiple frames which cause the user's swiped word to be prematurely cut off. This results in the user having to re-start their swipe, which is lessens the overall text entry speed. My system does have allowance for no fingers to be present in 5 frames before reasoning that the swipe has been finished, however the issue still persists in cases where detection fails for more than 5 frames. Increasing this window of allowance for failed detections would reduce in a lag when a user switches from a swipe mode 'on' gesture to a swipe mode 'off' gesture.

- While word gesture keyboards are well-suited to a full-arm swipe gesture in terms of intuition and learnability, the speed advantages that WGKs have exhibited on mobile platforms are lost when the physical area over which the user swipes exceeds a certain size.

Based on responses to the questionnaire, it is apparent that, though users did not find the system easy to use, most felt that the novel method for text-entry was enjoyable and fun. Users unanimously agreed that the system was intuitive to use - there is little prior knowledge required to use it for the first time - and also that learnability of the system was high, as its usage became easier after more time was spent with the system. Users mostly agreed that having to keep their arm outstretched while making swipe gestures across the keyboard region became tiring. This fatigue issue, known as "gorilla arm", acts as a further obstacle to usage. Users were generally satisfied that the system reacted appropriately to their finger movements, while opinion regarding the speed with which the system responded to their movements was split. This may be down to the fact that more tech-savy users would perceive the minor lag in the output display caused by the computational complexity of the image processing code functions. Most users did not appear to notice this slight lag. Users who had previously used word gesture keyboards for their phones performed better than those who hadn't, this may be related to the learning curve associated with WGKs themselves.

The learnability study, conducted for a single user, showed that the user's speed improved over time, in addition to their error rate falling(as the gap between gross and net WPM decreased). This indicates that overall performance of users could improve with practice and continued usage.

Overall, while the system has a number of limitations in terms of achievable speed and accuracy, it does function well as a new, novel interface system. I believe that the intuitiveness and learnability of the system are enhanced by its AR-based visualization approach. While it can't match the speed of physical keyboards and other existing touch typing systems, its main advantage is that it provides a way to enter textual data in mid-air, without ever needing to touch any physical surfaces, and hence it eliminates the risk of contracting germs and microbes through the hands. Furthermore, the speed and fatigue issues may be less of an issue if the keyboard is used in a public kiosk systems such as SureWash, or ATM machines, since the volume of text data entry requirements for such systems is typically low. Users tend to use these public computer systems only to enter data such as a username, or search keyword, rather than needing to write a document or essay.

A simplified model of the system was also developed following feedback from the user study. The system uses the same mechanism for touchless data input, but rather than provide keyboard/typing functionality, the system is intended for simpler tasks such as augmented reality questionnaires. In this experimental application of the system, the GUI displays a textual question with 4 virtual buttons. The buttons can either depict the answer as a picture, or can contain text. For example, a user may be asked to select from four choices, the button with an image of a human heart, while the other 3 choices depict other organs such as the brain, liver and lungs. This questionnare application was only tested informally, but due to the reduced number of buttons to select from (only 4 available buttons rather than >26), their increased size, and the space between them, accuracy of button selections was much greater and speed was also improved. This feature, among some other system aspects leave scope for future work in touchless and gestural interaction.

6.5 Future Work

There are numerous aspects of the system which could be explored and expanded upon in the future. Firstly, I feel that, while word gesture typing lends itself well to touchless hand gestures as a natural way to enter data, a two-handed touchless keyboard based around mid-air button presses may provide better performance in terms of speed. While simulating a button press in mid air may seem less natural and intuitive than a swipe gesture, the addition of a second hand in a gesture style that is more familiar to users, and more closely models traditional touch typing, could see benefits in performance and usability, while also retaining the touchless functionality of the system.

A more advanced swipe resolution algorithm would likely improve the error rate and Net WPM results exhibited of the system. By implementing the swipe resolution process by taking other metrics into account, such as the shape properties of the actual swipe curve itself, and also developing a string search approach which can more accurately infer the keys user's intended word from the intermediate keys or accidental keys, many of the systems existing problems can be solved.

Improved finger tracking algorithms which can account for temporary occlusions of fingertips could further reduce the keyboard's error rate by preventing failed detections from requiring users to re-type their word from the beginning. A predictive tracker such as the Kalman filter could potentially provide this need, if the hand's motion can be accurately modelled.

Finally, I believe that the system demonstrated the feasibility of computer vision, depth-based hand tracking, and augmented reality interface design as techniques which lend themselves to touchless typing and general gesture-based interfaces. Future work in the broad field of gestural interface design could also benefit from such approaches.

Chapter 7

Conclusion

In this paper, I developed a touchless, gesture-controlled keyboard for an existing interactive hand hygiene training system(SureWash). The system enables users to enter text by forming words with continuous swipe gestures in mid-air, providing a natural way to interface with computers using our hands rather than being constrained by physical keyboard components.

The keyboard was developed to be used with inexpensive depth sensing camera hardware. It combines computer vision with fuzzy string matching techniques to provide both image processing and word gesture typing functionality. The system also utilizes augmented reality to construct an interactive, real-time graphical user interface.

The system has several limitations. Typing speeds were generally quite slow, as the control of the top-down keyboard UI in mid-air was found by users to be somewhat difficult. This is due to the complexity required to accurately navigate between buttons using the hand's free motion in mid-air without any physical frame of reference. Furthermore, the system's error rate was exacerbated by the swipe resolution implementation used, which requires a high degree of accuracy in the user's swipe. These issues leave scope for future work in the domain. In spite of these limitations, users generally found this novel approach to swipe-based typing to be intuitive and enjoyable.

My work in this dissertation provides a potential solution to the need for touchless computer interaction in healthcare environments and demonstrates that touchless typing can be feasibly implemented with a word gesture keyboard.

Bibliography

[1] Alan Dix, Sandra Cairncross, Gilbert Cockton, Russell Beale, Robert St Amant, and Martha Hause. Human-computer interaction, 1993.

[2] John Canny. The future of human-computer interaction, 2006.

[3] Manju Kaushik and Rashmi Jain. Gesture based interaction NUI: an overview. *CoRR*, abs/1404.2364, 2014.

[4] Dana Alexander Brandon Savage, Mark Segal. The cost of healthcare associated infections, 2011.

[5] Jan Nash. More hospitals 'checking-in' to kiosks. http://blogs.zebra.com/blog/bid/48548/More-Hospitals-Checking-in-to-Kiosks, 2011.

[6] Greg Pond. The medical use of touchscreen technology is a magnet for bad bugs, 2014.

[7] Richard A. Bolt. &ldquo;put-that-there&rdquo;: Voice and gesture at the graphics interface. *SIGGRAPH Comput. Graph.*, 14(3):262–270, July 1980.

[8] James Davis and Mubarak Shah. Recognizing hand gestures. In *Computer Vision — ECCV '94*, pages 331–340. Springer-Verlag, 1994.

[9] Adiyan Mujibiya, Takashi Miyaki, and Jun Rekimoto. Anywhere touchtyping: Text input on arbitrary surface using depth sensing. In *Adjunct Proceedings of the 23rd Annual ACM Symposium on User Interface Software and Technology*, UIST '10, pages 443–444, New York, NY, USA, 2010. ACM.

[10] Johannes Roith, Tayfur Coskun, Teodora Velikova, Sebastian Klingenbeck, and Gudrun Klinker. Gestairboard: A gesture-based touch typing keyboard using

the kinect camera. In Gesellschaft für Informatik, editor, *Informatiktage 2013 Fachwissenschaftlicher Informatik-Kongress*, volume S-12 of *Lecture Notes in Informatics (LNI) - Seminars*, pages 137–140, Bonn, 2013. Köllen Druch+Verlag GmbH.

[11] Sebastiaan Michael Stuij. Usability evaluation of the kinect in aiding surgeon-computer interaction. Master's thesis, 2013.

[12] *Using Commodity Visual Gesture Recognition Technology to Replace or to Augment Touch Interfaces*, volume 15. University of Twente, 2011.

[13] Farzin Farhadi-Niaki, S. Ali Etemad, and Ali Arya. Design and usability analysis of gesture-based control for common desktop tasks. In *Proceedings of the 15th International Conference on Human-Computer Interaction: Interaction Modalities and Techniques - Volume Part IV*, HCI'13, pages 215–224, Berlin, Heidelberg, 2013. Springer-Verlag.

[14] Vladimir I. Pavlovic, Rajeev Sharma, and Thomas S. Huang. Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(7):677–695, July 1997.

[15] Dr.Anil Kumar Nandy Prof. Rajeshri Rahul Itkarkar. A study of vision based hand gesture recognition for human machine interaction, 2014.

[16] G. R. S. Murthy and R. S. Jadon. A review of vision based hand gestures recognition. *International Journal. Of Information Technology and Knowledge*, pages 2–2009.

[17] Shahriman AB Siti Khadijah Zaba Hazry Desa Mohd Azri Abd Aziz Nazrul Hamizi Adnan, Khairunizam Wan. The development of a low cost data glove by using flexible bend sensor for resistive interfaces. 2012.

[18] Ronald T. Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, August 1997.

[19] Shumin Zhai and Per-Ola Kristensson. Shorthand writing on stylus keyboard. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 97–104. ACM, 2003.

[20] Shumin Zhai and Per Ola Kristensson. The word-gesture keyboard: Reimagining keyboard interaction. *Commun. ACM*, 55(9):91–101, September 2012.

[21] Michael S. Lasky. Review: Celluon magic cube projection keyboard. `http://www.wired.com/2011/10/celluon/`, 2011.

[22] New York Times. Apples ipad tablet - slideshow. `http://www.nytimes.com/slideshow/2010/01/27/technology/01282010-apple-slideshow_7.html`, 2010.

[23] Jun Hongo. Nec brings augmented-reality keyboard to users forearm. `http://blogs.wsj.com/japanrealtime/2015/11/06/nec-brings-augmented-reality-keyboard-to-users-forearm/`, 2015.

[24] Kaist. K-glass 3 offers users a keyboard to type text. `http://www.eurekalert.org/pub_releases/2016-02/tkai-k3o022616.php`, 2016.

[25] Paul M Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental psychology*, 47(6):381, 1954.

[26] Ville Mäkelä, Tomi Heimonen, and Markku Turunen. Magnetic cursor: Improving target selection in freehand pointing interfaces. In *Proceedings of The International Symposium on Pervasive Displays*, page 112. ACM, 2014.

[27] Aileen Worden, Nef Walker, Krishna Bharat, and Scott Hudson. Making computers easier for older adults to use: area cursors and sticky icons. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, pages 266–271. ACM, 1997.

[28] G. Bradski. Opencv. *Dr. Dobb's Journal of Software Tools*, 2000.

[29] Kenneth Dawson-Howe. *A practical introduction to computer vision with OpenCV*. John Wiley & Sons, 2014.

[30] Suzuki, S. and Abe, K. Topological structural analysis of digitized binary image by border following. *Computer Vision, Graphics and Image Processing*, 30(1):32–46, January 1985.

[31] Jakub Segen and Senthil Kumar. Human-computer interaction using gesture recognition and 3d hand tracking. In *Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on*, pages 188–192. IEEE, 1998.

[32] Tola Chhoeun. How does swipe keyboard work? `http://tola-chhoeun.weebly.com/tech-blog/how-swipe-keyboard-works`, 2014.

[33] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, February 1966.

[34] M. Gilleland. Levenshtein distance, in three flavors. `http://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Fall2006/Assignments/editdistance/Levenshtein\%20Distance.htm`, 2016.

[35] Chen-Chiung Hsieh, Dung-Hua Liou, and Wei-Ru Lai. Enhanced face-based adaptive skin color model. *Journal of Applied Science and Engineering*, 15(2):167–176, 2012.

[36] Yikai Fang, Kongqiao Wang, Jian Cheng, and Hanqing Lu. A real-time hand gesture recognition method. In *Multimedia and Expo, 2007 IEEE International Conference on*, pages 995–998. IEEE, 2007.

[37] Benjamin D Zarit, Boaz J Super, and Francis KH Quek. Comparison of five color models in skin pixel classification. In *Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, 1999. Proceedings. International Workshop on*, pages 58–63. IEEE, 1999.

[38] Ahmed Sabbir Arif and Wolfgang Stuerzlinger. Analysis of text entry performance metrics. In *Science and Technology for Humanity (TIC-STH), 2009 IEEE Toronto International Conference*, pages 100–105. IEEE, 2009.