FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

A framework to adapt WIMP interfaces to No-Touch

João da Silva Fernandes

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Jorge Alves da Silva

July 10, 2014

A framework to adapt WIMP interfaces to No-Touch

João da Silva Fernandes

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: José Manuel de Magalhães Cruz (Assistant Professor at Faculdade de Engenharia da Universidade do Porto)

External Examiner: Teresa Isabel Lopes Romão (Assistant Professor at Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa)

Supervisor: Jorge Alves da Silva (Assistant Professor at Faculdade de Engenharia da Universidade do Porto)

July 10, 2014

Abstract

The introduction of technology in hospitals resulted in an increase of health-care associated infections, as doctors would use poorly sterilized computers before interacting with the patient. There are workarounds to these infections, such as better sterilization procedures for technology, which is still imperfect, sterilization procedures for the doctor, which is time consuming and use of an assistant to interact with he computer at the doctor's orders, which is also time consuming and can be frustrating for the doctor. No-touch interfaces provide an opportunity to solve these infections. However, resources for developers who wish to implement these interfaces are scarce, with science fiction movies such as Minority Report still acting as one of the main guidelines for interaction design.

A collaboration was made with Glintt HS to understand the interactions required by doctors in the operating room, using Glintt's eResults application as a study case for the development of a framework to easily adapt any web-based interface to a no-touch interface. This application allows doctors to browse through the patient's clinical history and view results, which may be documents, images or videos.

Literature in this area shows that there are very few guidelines for the development of no-touch interfaces and the only existing frameworks are simple mouse replacements with no impact on the interface itself. Regarding the operating room, the Microsoft Kinect has been successfully used for image manipulation in the past two years. The technological alternatives are standard RGB cameras and the LeapMotion, but the Microsoft Kinect proves itself more adequate as it possesses a depth sensor and a maximum range of 4 meters.

A web framework was developed that allowed developers to create the no-touch interfaces by merely adding the classes offered by the framework to the HTML elements they wish to make interactive via no-touch. Furthermore, the framework offers an API that can receive JSON messages representing input from a no-touch device and processes the interaction, and a native program was made to pass the Kinect's tracking data to the API.

Development followed a Rapid Iterative Testing and Evaluation (RITE) methodology, where various interactions were implemented and tested with one or two users, for each of the various interaction components, such as button pressing, list scrolling or image rotation. After each component had satisfactory interactions, a no-touch prototype of Glintt's eResults was made and, finally, larger usability tests were done on this prototype to measure the efficiency of the adaptation.

Resumo

A introdução de tecnologia nos hospitais veio acompanhada de um aumento de infeções relacionadas com cuidados de saúde, uma vez que os médicos utilizam computadores mal esterilizados antes de interagir com o paciente. Existem formas de contornar estas infeções, tais como melhorar os procedimentos desinfeção do equipamento, que ainda mostram várias falhas, procedimentos de desinfeção do médico, que consome bastante tempo, e o uso de um assistente para interagir com os computadores seguindo as ordens do médico,, que também é pouco eficiente a nível de tempo e pode ser frustrante para o médico. As interfaces no-touch ofrecem uma oportunidade para resolver este problema. No entanto, os recursos para desenvolver estas interfaces são escassos, sendo que filmes de ficção científica como o Minority Report ainda são dos melhores pontos de referência disponíveis para desenhar a interação.

Foi feita uma colaboração com a Glintt HS, no sentido de perceber quais os casos de uso que os médicos necessitam numa cirurgia, utilizando a aplicação eResults da Glintt como o caso de estudo para o desenvolvimento de uma framework para adaptar facilmente qualquer interface web para uma interface no-touch que utiliza tecnologias web. Esta aplicação permite aos médicos navegar pelo historial clínico e consultar resultados, que podem ser documentos, imagens ou vídeos.

A literatura nesta área revela que existem poucas linhas de apoio ao desenvolvimento de interfaces sem-toque e, no que diz respeito ao seu uso na sala de operações, o Microsoft Kinect tem sido usado com sucesso na manipulação de imagens nos últimos dois anos. As alternativas tecnologicas são câmeras RGB comuns e o LeapMotion. No enanto, o Microsoft Kinect revela-se mais adequado, uma vez que possui um sensor de profunidade e um alcance máximo de 4 metros.

Foi desenvolvida uma framework web que permite a criação de interfaces no-touch, adicionando apenas as classes oferecidas pela framework aos elementos HTML que se pertende tornar interactivos via no-touch. A framework oferece ainda uma API que recebe mensagens JSON representando movimento capturado por um dispositivo para interação no-touch e processa a interação e foi desenvolvido um programa nativo para passar os dados capturados pelo Kinect para a API.

O desenvolvimento seguiu uma metodologia RITE, onde as várias interações foram implementadas e testadas com um ou dois utilizadores para cada um dos componentes, tais como a seleção de elementos e manipulaçãod e imagens. Quando todos os componentes tiveram interações satisfatórias, foi criado um protótipo de uma versão no-touch do eResults da Glintt e, no final, foram realizados testes de usabilidade numa escala maior para medir a eficiência da adaptação.

Acknowledgements

I would obviously like to thank my parents for giving me all the opportunities and support they could to get me to this point. They may not be perfect, but who is? They always did what they believed was best for me, often casting aside their own convenience and I am most thankful for that.

I would also like to thank all my friends who I met during my path through university, both those that I befriended earlier and later, those that accompanied me through bad times and good times. They are like a family to me and I hope to keep them by my side during my future. I also wish them best of luck during the future, both for those that are about to start their career and those that still have to endure a few more years before they reach the point where they have to write a document like this one. I also have to thank the people responsible to unite us by maintaining a centenary tradition, despite all the hardships that I now know come associated with it.

I should also thank all professors from my academic path, all the way back to the grade school professor that taught me how to read and calculate. I learned lessons from all of them that extend beyond the academic materials they taught, either from their virtues or their flaws. In particular, I would like to thank professor Jorge Silva for accepting to supervise me and showing concern with my work and being very helpful throughout both this semester and the preparation semester.

Additionally, I would also like to thank Pedro Rocha from Glintt for giving me the opportunity to work on this thesis and my supervisor at Glintt, Pedro Pinto, for continuously discouraging me from taking the easiest path, which made me create better solutions to the problems that arose.

Finally, I would like to thank my friend and brother Eduardo Jesus, who passed away earlier this year. A man of outstanding intellect and moral integrity, he was a major help throughout my academic path on this course and a grim reminder that one should take care of himself and not give in to the pessimistic side of the mind. May he rest in peace.

João Fernandes

"Watch your thoughts for they become words,
watch your words for they become actions,
watch your actions, for they become habits,
watch your habits for they become your character,
watch your character for it becomes your destiny."

Unknown author

Contents

List of Figures

LIST OF FIGURES

List of Tables

Abbreviations

HAI	Health-care Associated Infection
RGB	Red Green Blue
FPS	Frames Per Second
API	Application Programming Interface
IR	Infrared
LED	Light Emitting Diode
SDK	Software Development Kit
ROI	Region Of Interest
HCI	Human-Computer Interaction
WIMP	Windows, Icons, Menus, Pointer
NUI	Natural User Interface
SDK	Software Development Kit
DOM	Document Object Model
RITE	Rapid Iterative Testing and Evaluation

Chapter 1

Introduction

This thesis, called no-touch interfaces, consists in the definition of rules for the development of no-touch interfaces and the implementation of a framework to ease the development of such interfaces, using a clinical history application as the case study. It was proposed by Glintt Healthcare Solutions and the work was conducted at this same company. This document details the proposed problem, the existing work related to this topic and the implemented solution, as well as the results derived from this solution.

1.1 Context

With the introduction of computer technology in hospitals, the number of HAIs (Health-care Associated Infections) has been increasing due to poor sterilization methods of this technology. In the USA alone there were approximately 1.7 million HAIs in 2007 [RB12]. One environment that is particularly prone to these occurrences is the operating room, where a doctor may need to use a computer to obtain more information about the patient. However, not only is this prone to an HAI, but it is very time costly, as the sterilization processes may add up to 2 hours to an operation [Bau11]. One possible solution is the use of an assistant to interact with the computer in the way desired by the doctor. Nevertheless, this solution proves itself ineffective as it is prone to mistakes and still cost ineffective, where in an extreme case, a simple instruction can take several minutes to be executed [JOS+11].

1.2 Problem to solve

With the advent of inexpensive technology for no-touch input, such as the Microsoft Kinect and the LeapMotion, there is an opportunity to solve the problem of computer interaction in the operating room. However, developing applications using these interfaces is still troublesome as there are only a few guidelines available and the only existing framework is exclusive to native applications

using the Microsoft Kinect, which is particularly limiting with the increasing popularity of web applications. Therefore, the objectives of this thesis are the definition of a set of rules of the best interaction methods for no-touch interfaces and the implementation a framework that allows the developers to skip on the interaction design and implementation.

1.3 Objectives

The objectives of this thesis are as follows:

1. Find the best methods to apply no-touch interfaces in a series of use cases defined by Glintt Healthcare Solutions as necessary for a doctor during surgery;
2. Implement a Javascript framework implementing these methods, so that future developers can develop applications with no-touch interfaces by merely importing the framework's stylesheet and scripts, adding the appropriate classes to the HTML and initializing the framework;
3. Create a prototype with a no-touch interface, using the developed framework, of Glintt's eResults application, which allows a medic to view a patient's clinical history;
4. Execute usability studies to verify whether the adaptation was successful.

1.4 Expected benefits

Upon successful completion of this thesis there will be gains for both doctors (and by extension their patients) and developers. Doctors will be able to access information during surgeries, faster while being able to avoid HAIs, resulting in overall better healthcare. Developers will also save time by skipping the interaction design portion of the application development and by using the developed framework.

1.5 Document structure

Besides the introduction, this document contains four more chapters. In chapter 2, the problems to be solved are explored in further depth and Glintt's eResults application is detailed. Afterwards, in chapter 3, there is a presentation of the technology and research publications available in this area. In chapter 4 the implemented framework is described and examined, both from technical and usability viewpoints, the eResults adaptation is exposed along with the challenges that arose despite the aid from the framework. Finally, in chapter 5 there is a small review of the contents discussed in this thesis, a discussion of the results and a listing of possible improvements that fell out of the scope of this work.

Chapter 2

Problem description

As previously mentioned, this thesis intends to discover the best way to translate components of WIMP (Windows Icons Menus Pointer) interfaces into no-touch interfaces, which requires both the creation of a gesture vocabulary and rules for element display. This chapter will describe the challenges that arise upon the development of a gesture vocabulary for no-touch interfaces, followed by a description of the components required by eResults and should therefore be adapted to a no-touch interface and concluding with a description of the additional constraints required for this application.

2.1 Gesture Vocabulary

No-touch interfaces are a subgroup of Natural User Interfaces(NUIs) and, as such, it was necessary to define a gesture vocabulary for all the implemented components, which refers to the set of gestures used to interact with the application. This set must take the following issues into account:

- False positives, where the system recognizes a gesture that the user didn't preform, should be avoided. This kind of errors causes the system to react without the users understanding the reason for this, leaving them confused and having a negative impact in the user experience. If there is a risk of false positives, the users should be able to easily undo the action.

- False negatives, where the system fails to recognize a gesture preformed by the user, should be avoided as well. This kind of errors make the user lose confidence in the system but since they do not change the application state, they won't have a big impact in the user experience if they are not frequent and the users are able to correct the error and are aware of what they must do in order to correctly perform the gesture.

- Gesture vocabulary collision, where similar gestures have different results, should also be avoided, since this is a driving force to cause false positives.

- The user should also feel comfortable using the gesture vocabulary. Ideally, this translates into being able to guess the gestures, but, otherwise, the user required action should feel natural to the user. As Wixon and Wigdor [WW11] mention, the "Natural User" and "Interface" parts in NUI should be separated, rather than "Natural" and "User Interface".
- The gesture vocabulary should not be tiresome, in order to allow the users to interact with the system for long periods.

2.2 The eResults components

The eResults application is the study case for this thesis, as it has the necessary features for doctors in the operating room, and, therefore, it defines the components that need to be adapted. The most basic component to adapt, and one that should be transversal to all applications is element selection. This encompasses buttons, checkboxes, list elements and anything else that would be clickable in a WIMP interface. In addition to this basic element, the following components were established as necessary by Glintt, as they either exist in the current eResults platform:

- The eResults platform uses lists of elements either occupying the page's full width or a smaller rectangle. Figure 2.1 shows a listing for results of a given patient, along with the filters associated with it. However, the implemented framework should be able to work with lists of any style and number of elements, which brings up questions in how to space the elements and how to scroll through the list;

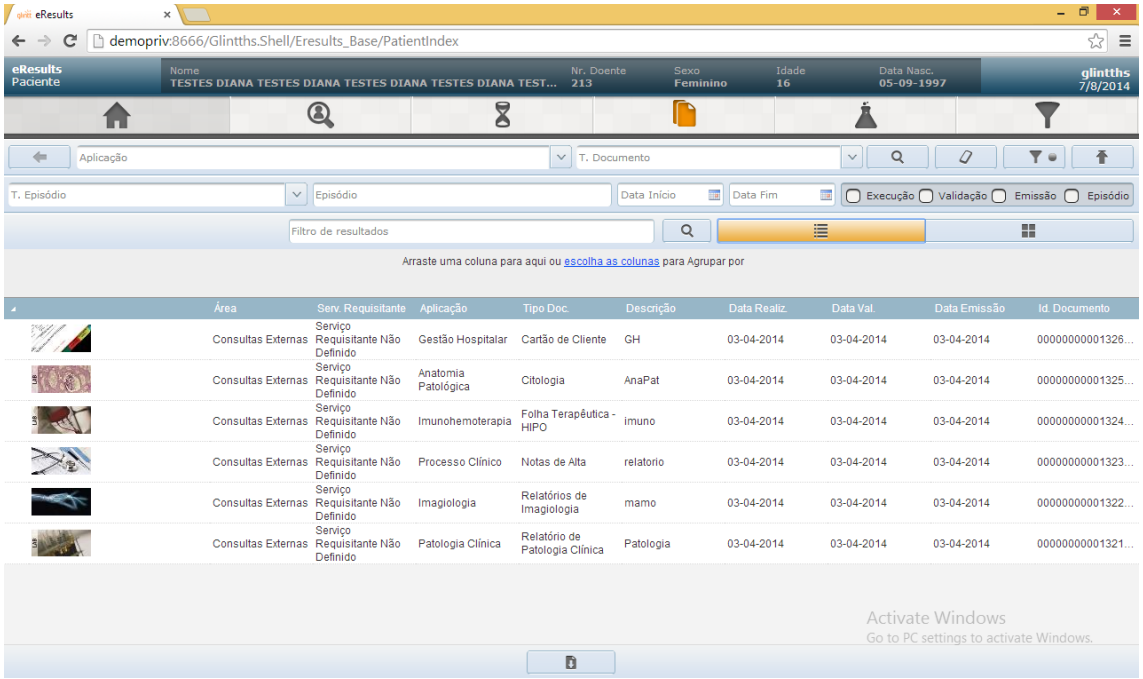

Figure 2.1: Results listing in eResults.

- These lists should be able to be filtered, which is achieved in eResults via drop-down lists, as seen in figure 2.1. These drop-down lists can have as little as 5 and as many as 500 elements and may contain a whole HTML div, rather than simple text. Under these conditions, it would be naive to assume that it is sufficient to drop down the elements in a list, having only applied the same changes as the ones for a regular list;

- The patient records in eResults may be images representing, for example, x-rays or snapshots of an ecography, as seen in figure 2.2, which the user should be able to view, as well as move, rotate and zoom in and out. It is necessary to determine the best way to apply these transformations with gestures;

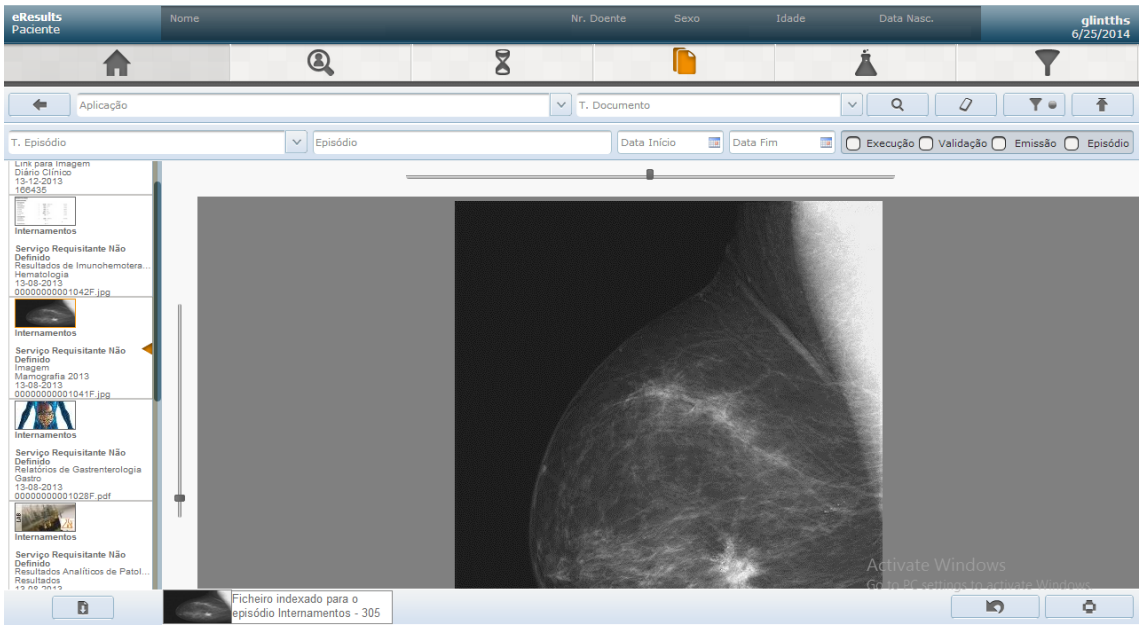

Figure 2.2: Sample image result in eResults.

- There may also be records pertaining to documents regarding the patient, such as analysis or certificates passed by a doctor, as seen in figure 2.3, which shows an example of a document inserted in eResults for testing purposes. These documents are stored as PDF files, where the user should be able to scroll horizontally and vertically, as well as zoom in and out, identically to images;

- Although eResults does not currently support video elements, Glintt established that his feature could be desirable in the near future, in order to allow the users to review previous interventions on the patient, surgical or otherwise. In this type of results, the user should be able to pause or play the video at will, as well as seek within the video, in order to see what happens at a certain point within it.

Additionally, eResults also makes use of text introduction in order to filter through results. Nevertheless, several authors recommend a multimodal approach to this problem, using speech

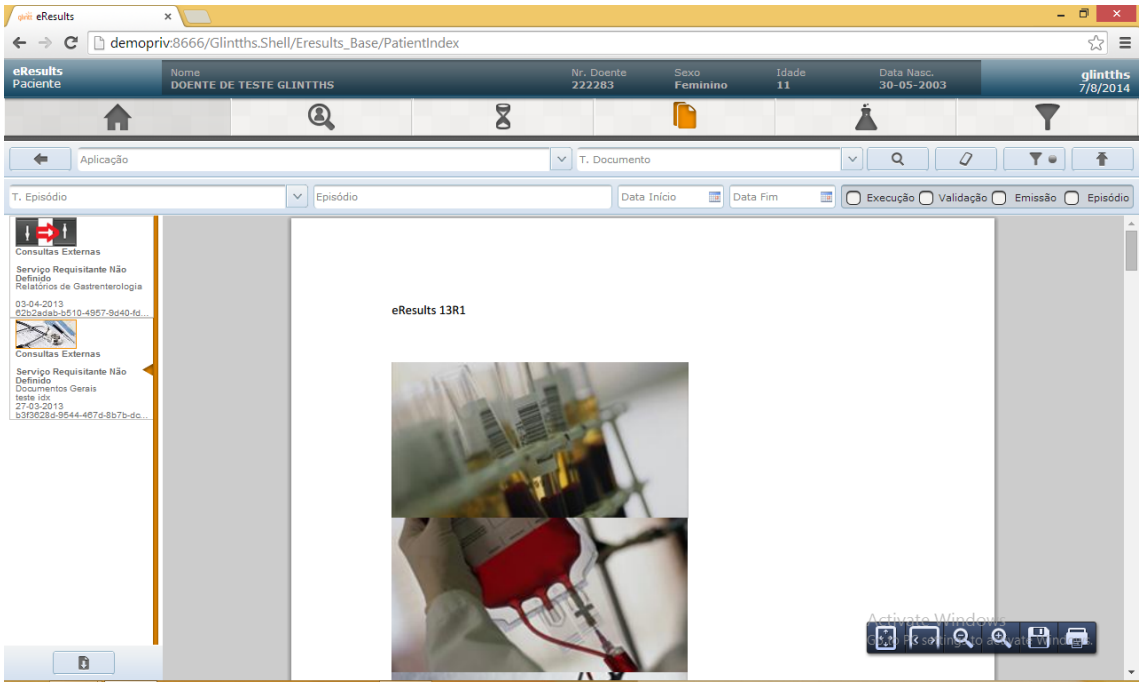

Figure 2.3: Sample document in eResults.

recognition as an auxiliary modality. This would not be a favourable solution for text input in eResults, since much of the text introduction refers to medical and pharmaceutical terms and products, most of which are not found in common voice recognition dictionaries. Although gesture based keyboards were considered, making them usable without being tiring would prove itself a much bigger challenge that would fall out of the scope of this thesis.

2.3 Interaction constraints

All of the defined interaction and gesture vocabulary should also take into account the following constraints:

- The user should be able to interact with the system at a distance of three meters. This means that the chosen technology should correctly identify user motion and state at this distance;
- There should be a procedure for the users to signal their intention to start interacting with the system, in order to avoid false positives when the users are moving their hands as part of a conversation or interaction with the environment;
- The system should also be able to assert when the user has stopped interacting with it, either by an explicit user action signalling the disengagement or by a contextual assertion from the system. This thesis should also answer which of these is the best approach to the problem.

Chapter 3

State of the art

This chapter will start by further specifying the context of the problem. Then, a literature review follows, explaining what has been done within this domain. Finally, there is an overview of the possible technology options and a decision regarding which one to use.

3.1 Intervention domain

This thesis belongs to the HCI (Human-Computer Interaction) domain, more specifically to the sub-domains of medical computer interaction and NUIs and,within this sub-domain, the area of gesture interaction. More precisely, there are problems in medical computer interaction that may be solved through gesture interaction while extracting lessons for the gesture interaction scientific community and developing a framework that implements these lessons.

3.2 Existing works

The existing works can be divided into three main areas: the research done attempting to find universal gesture sets, the documents that already help developers creating no-touch interfaces and the work done in the use of no-touch interfaces in surgery.

3.2.1 Gesture research

One important work when studying this area is Karam and Schraefel's taxonomy of gestures [Ks05]. These authors did a revision of the literature about gesture interaction and divided gesture interaction into 4 types:

- Deictic Gestures - Gestures that involve pointing to identify the location of an object;
- Manipulative Gestures - Gestures that mimic interaction with a physical object, such as grabbing;

- Symbolic Gestures - Gestures where the user controls the system by performing a gesture or a pose with implicit meaning, usually of cultural significance, such as telling it to stop by holding the hand with the palm facing forward;

- Gesticulation - Reading of gestures during verbal communication in order to better identify the user's intention.

This taxonomy helps identifying the most adequate type of gestures for a given context. For example, for a situation that mimics a real world situation, manipulative gestures would be more fitting, while for giving instructions, symbolic gestures might be more adequate.

There are also a number of works trying to find some sort of consensus regarding optimal gesture interaction. Notably, Nielsen et al., in 2003 claimed that it is impossible to find an universal gesture vocabulary [NSMG03]. Wachs et al. made similar claims in 2008 [SWE08]. Although this may be true, their experiences are mostly done with symbolic gestures, which are more prone to cultural differences.

Contrary to these claims, Shedroff and Noesel [SN12] have studied science fiction movies that feature no-touch interfaces, such as Minority Report, and found patterns from gestures their characters use to interact with their interfaces. They assume that if these gestures are commonly used in science fiction, it is because all their creators found them intuitive, which may be representative of what users will also find intuitive. They make the following conclusions relevant to this thesis:

- Physical analogies are always used when possible, such as pushing to move or turning the hands to rotate the object. This means that the previously described manipulative gestures are common in these interfaces;

- Objects should have inertia similar to their physical counterparts and should move according to the force applied in the gesture;

- Since there is no physical action to scale an object, grabbing an object and spreading the hands tends to make them larger, while bringing the hands together tends to make them smaller, as these actions are equivalent to stretching or compressing an object;

- Waving is commonly used to activate the interface or system, such as waving other a water faucet to start the water;

- Swiping in either horizontal direction is commonly used to dismiss objects;

- Pointing and touching the objects in the air is commonly used to select them;

Additionally, in no-touch interfaces, the sensor, or input method, is always on, which needs to be addressed in order to prevent false positives, without making the gesture vocabulary so convoluted that false negatives are frequent. Wixon and Wigdor [WW11] address this issue by suggesting the use of a clutch analogous that acts as an analogous action to touching the screen or mouse and Chen [Che14] reinforces this suggestion by using the All The Cooks application

for the LeapMotion as an example and also suggests cooldown periods for actions that cannot use a clutch, such as a swipe, in order to prevent false positives when the user is returning to the base state. Additionally, both these authors and Microsoft [Mic13b] put a heavy emphasis in the iteratively testing prototypes and alternative gesture vocabularies and interfaces with users, followed by engineering of a new iteration, as opposed to the definition of a gesture vocabulary with statistic basis on user suggestions.

3.2.2 Developer Guidelines

There exist surprisingly few guidelines for the development of no-touch interfaces and they are mainly offered by the companies that provide the technologies for no-touch interfaces. One of these guidelines is offered by LeapMotion, the company responsible for the development of the no-touch technology of the same name, that provide guidelines for menus and user experience in general. The user experience guidelines [Lea13b] are vague and offer common-sense rules for the development of no-touch interfaces. However, the menu design guidelines [Lea13a] offer some specific alternatives for the creation of a menu to be used with the LeapMotion, by recommending touch emulation using a touch zone and using grid or radial menus. Although these rules are made with the LeapMotion in mind, they should port fairly well to any technology used for a no-touch interface and can be used in this work, for example, to filter results.

The Microsoft Kinect User Interface Guidelines [Mic13b] is a one size fits all document that gives usability recommendations for developers intending to use the Kinect. Table 3.1 comes from these guidelines and it recommends button dimensions for different screen resolutions. The 1080p resolution is highlighted, as the Kinect interface elements provided by Microsoft were built for this resolution. This document also gives a specific recommendation for targeting and selecting a button, by recommending hovering the hand and pushing it towards the screen, and for starting interacting with the application while avoiding false positives. However, outside of these specific situations, it mostly gives generic usability recommendations, such as the use of brighter colors for feedback and reminding the developer to keep in mind the environment in which the application will be used.

Overall, these guidelines are a good base for this thesis, but they need to be built upon for developers to be able to directly produce an application with a no-touch interface by merely following a set of rules, minimizing the need to consider the gestures and layouts to be used.

3.2.3 No-touch interfaces in the operating room

During the past decade there have been attempts at solving the sterile computer interaction problem, with the usage of conventional cameras and even pedals [GFG+04][WSE+07]. However, these saw little exploration due to the technological limitations at the time.

In the last three years there has been some research on the usage of the Kinect in surgery. Galo et al. developed a hand posture recognition system for use in 3D medical visualization that successfully recognized the user's actions with an accuracy of 73.53% using 10-fold cross-validation

Sceen Type	Screen Width(px)	Screen Height(px)	Button dimensions(px)
UHDTV	7680	4320	880
(W)QHD	2560	1440	294
1080p	1920	1080	220
WSXGA+	1680	1050	214
HD+	1600	900	184
WXGA+	1440	900	184
WXGA	1366	768	157
720p	1280	720	147
XGA	1024	768	157
SD	720	480	98
HVGA	480	320	66

Table 3.1: The recommended button dimensions for use with the Microsoft Kinect

on distances up to 2 meters[GPP12], and in another study they suggest a 2-hand approach for interaction[GPC11]. Table 3.2 lists the gestures proposed in this later work, where the Kinect is used for navigating a 3D image and selecting a single ROI (region of interest).

	Static posture recognition	Dynamic gesture recognition
Pointing - point	ONLY the DOMINANT hand is active	none
Pointing - click	BOTH hands are active (while already in pointing state	palm of the NON-DOMINANT hand closed-open-closed (sequence)
ROI extraction	folded arms	none
ROI erasing	ONLY the DOMINANT hand is active	unstable movements for 500ms
Animating	ONLY the NON-DOMINANT hand is active	none
Zoom	BOTH hands are active AND palms facing forward	discordant movements in the XY plane
Translation	BOTH hands are active AND palms facing forward	concordant movements in the XY plane
Windowing	BOTH hands are active AND one palm facing forward, the other with a clenched fist	none
Rotation	BOTH hands are active AND with clenched fists	none

Table 3.2: Gestures proposed by Galo et al. for image interaction

Juhnke [Juh13] refers to studies done in the area of gesture interaction for the choice of gestures, but never reveals what those selected gestures were. The results in her study do, however, suggest that the Kinect can achieve better interaction times, at an accuracy of 85%.

Wachs et al. [JWP12] [JW13] did research on context-based gesture interaction, where the computer tries to identify the intention of the user by taking contextual cues that supplement the

gestures. Although this seems like an interesting approach, it lies within the area of artificial intelligence and, therefore, is out of the scope of this thesis. Nevertheless, he suggests a series of gestures, as seen in Figure 3.1 which still achieve over 90% accuracy when used without a contextual approach.

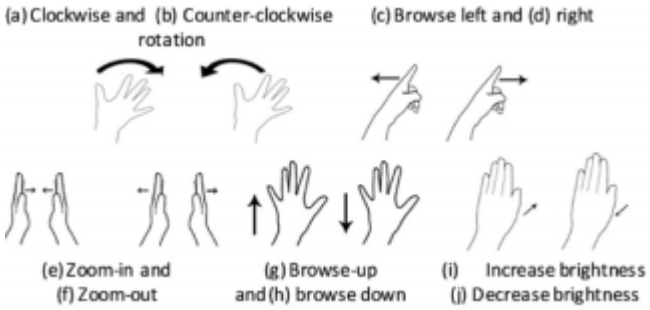

Figure 3.1: Gestures proposed by Wachs et al. for image interaction

All of these works take only image manipulation into account and, therefore, are insufficient to cover all cases needed for Glintt HS's clinical history application. Nevertheless, Galo and Wach's gesture proposals will be taken into account when proposing gestures for image and document manipulation.

3.3 Technology

The choice of hardware for gesture recognition is critical in this work, as each one has unique characteristics that define limitations of the gesture precision precision and range.

3.3.1 Existing options

When discussing technology for no-touch interaction, there are three main possibilities: RGB (Red Green Blue) cameras, the Microsoft Kinect and the LeapMotion.

RGB cameras allow the capture of images, which can then be analysed via computer vision, by employing techniques such as Hidden Markov Models or Naive Bayes' Classifier [HKR+10], to recognize a human shape and gestures. Then, methods such as machine learning can be used to translate a sequence of human shapes into gestures and there are multiple open-source APIs (Application Programming Interface) that already implement these methods.

The LeapMotion is a small rectangular device as seen in the Figure 3.2, equipped with three IR (Infrared) LEDs (Light-emitting diodes) and two IR cameras. These are facing upwards and the cameras capture IR reflections which are used by the device to perform calculations that accurately determine the hand and finger positions[Gor12]. It also has an API that allows the developer to obtain the finger and hand positions and detect a few predefined gestures [LM13].

The Microsoft Kinect is a device containing a RGB camera that can record at 30 FPS (Frames per second), which is an acceptable rate to uniquely identify gestures that takes longer than 200ms

Figure 3.2: The Leap Motion

to perform, since in that time it can capture 6 frames. It also contains an IR emitter and IR sensor, a tilt motor and a microphone array [Mic12a], as seen in the Figure 3.3. In practice, this allows for the functionalities of an RGB camera with the added possibility of depth mapping, thanks to the IR emitter and sensor. The tilt motor allows for adjustments depending on the user's height and the microphone array is used for voice recognition. There are several SDK for the Microsoft Kinect, including Microsoft's own Kinect for Windows [Mic13c] and open-source alternatives [Ope12a][Ope12b].

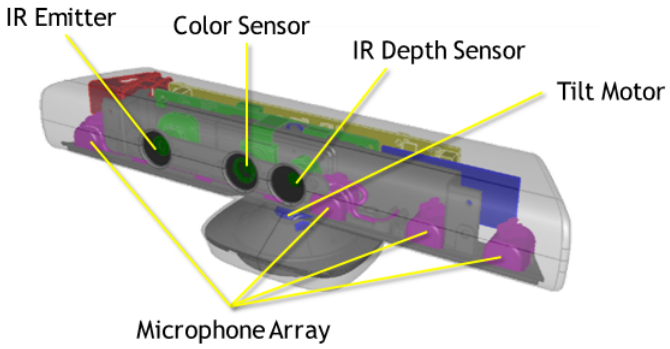

Figure 3.3: The Microsoft Kinect's components

3.3.2 Choice of technology

For this thesis the Microsoft Kinect will be used, since the depth mapping provided by the Kinect allows for an extra degree of freedom in user gestures, by allowing the users to approach their hand as an input method. It is also able to capture image and depth up to a range of 4 meters [Mic12b], as opposed to the LeapMotion, which can only capture information in a radius of 1 meter around it and must be placed below the target, limitations which are not desirable for surgery.

3.3.3 Existing SDKs

There are two SDKs available for the Microsoft Kinect: the official Microsoft SDK and OpenNI. OpenNI is an all-purpose SDK for body tracking and natural interaction, capable of working with any depth camera[Str14]. PrimeSense, the company behind the motion-sensing technology of the Kinect, is one of the main members behind OpenNI and developed a middleware for it, called NITE, to enable development for the Kinect with OpenNI [Fai12]. The main advantage of this framework over the Microsoft Kinect is the in-built gesture recognition, the automatic calibration offered and the wider array of programming languages that it supports, since it has wrappers for Java and Python. Its disavantages are the lack of documentation and support, as well as difficulty of instalation.

The Microsoft SDK, on the other hand, features numerous tutorials and samples, along with satisfactory documentation. This SDK is exclusive to Windows operating systems and the C++ and C# languages. It features body tracking and as of version 1.7, it offers a KinectInteraction API meant to facilitate the development of applications where the Kinect is used as the primary mean of interaction [Mic13d]. This API provides refined hand tracking, by being able to detect whether the hand is gripped or open and by being able to detect a press action and map the hand coordinates according to a physical interaction zone. Figure 3.4 shows Microsoft's description of the physical interaction zone, and it can be seen that it consists of two curved areas, one for each hand, in which interaction is considered comfortable. For each of these areas, the coordinates provided by the API should range from 0, at the leftmost or top points of the zone, to 1, at the rightmost or bottom points of the zone. However, in practice, the values frequently escape this range and the programmer must bind these values in the range or ignore values outside the range.

Another drawback of this implementation is that the hand state is lost if it leaves the physical interaction zone, that is, for example, if the right hand is closed and the user moves it all the way to the left shoulder, the hand will be considered open and there is nothing the user or programmer can do until the hand is returned to the physical interaction zone. As such, the hand should never be show on the screen if it is outside the physical interaction zone, in order to prevent incorrect feedback. It should also be noted that this system is designed for the user to interact with only a single hand. Finally, the KinectInteraction API also provides UI components, optimized for interaction with the Kinect. However, these are meant only for native applications and therefore unusable in this work.

The Microsoft SDK was chosen for this work because its documentation is more extensive than OpenNI's and because the KinectInteraction API eases the mapping of the hand position to screen coordinates and its grip and press detection enable many more possible interactions. However, gestures such as swipes and waves, which are already implementes on OpenNI, would have to be implemented manually.

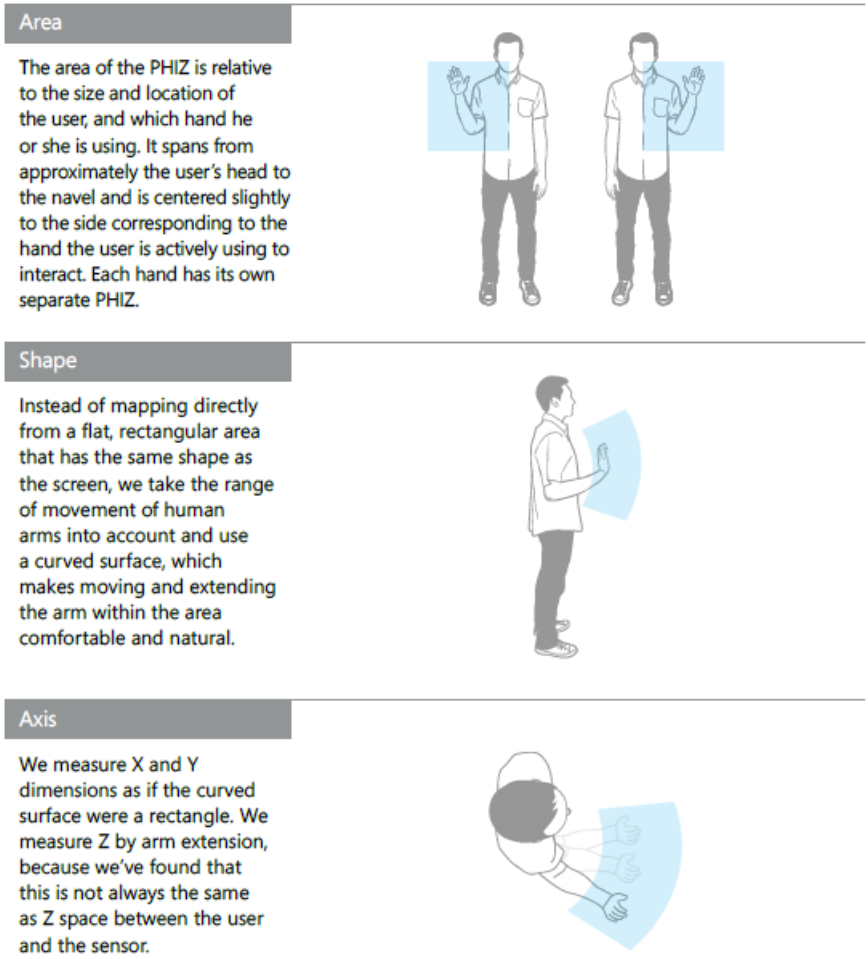

Figure 3.4: The physical interaction zone, as described by Microsoft

3.3.4 Web frameworks

The Microsoft Kinect cannot natively comunicate with webpages and, as such, there was a need to address this issue.

Microsoft offers a solution, as of Kinect SDK 1.8, which handles the communication, presents the hand cursors and includes pre-defined UI components to interact with the Kinect [Mic13e]. Although an attempt was made to use this API for the thesis, since this API allows the Javascript to fully control the Kinect and receive data from all its streams, it requires extensive setup for any page and performs poorly: the sample webpage provided takes well over 300ms to respond to any movement. In conclusion, a more simplified and efficient API is needed.

Another available solution is KinectJS [Kal12a], which proves itself to be more efficient and easier to set up. However, this solution is based on the Kinect SDK 1.0 [Kal12b] and has not been updated since 2012, which results in more jitter, worse support for arm extension and a complete lack of support for hand grip. These features open a lot of possibilities for interaction, which caused this framework to be discarded.

Finally, Vangos Pterneas presents instructions to implement Kinect-javascript communications based on websockets and the C# Fleck library [Pte13]. The sample project has no noticeable lag and the usage of JSON for messages makes it simple to extend the framework by including an operation type in every message. As such, the communication between the webpage and the Microsoft Kinect was implemented following Vangos Pterneas's tutorial.

3.3.5 Gesture recognition

The Kinect SDK together with the KinectInteraction API offer the possibility to track the position of multiple skeleton joints and whether the hand is gripped or not. However, it was also necessary to identify gestures, such as swipes or waves and gesture recognition libraries were investigated in order to simplify the implementation work and focus the thesis work on the gesture vocabulary design and usability. However, the existing libraries proved themselves to be hindering to the work: Kinect Dynamic Time Warping [Ste11] and Kinect Toolbox [Cat12], which made use of machine learning techniques to identify a performed gesture out of a set of pre-recorded gestures, were built upon older versions of the Kinect SDK, while GesturePak requires a paid license and is not clear regarding the computability with the latest SDK. Since these external libraries were not viable, gesture recognition was implemented using only the skeletal tracking provided by the Kinect SDK.

3.4 Usability testing

Since the goal of this work was to define and implement usability guidelines for no-touch interfaces, it is natural that usability testing would be performed with users in order to assess its success.

Regarding the moderation of usability testing, which refers to the interaction between the user and the person conducting the experiment, Bergstrom [Ber13] suggests four different techniques:

- In concurrent think aloud, users are asked to to report their thought process throughout the test session, in order for the moderator to obtain more precise feedback. However, this technique may interfere in usability metrics, as users may take more time to perform the actions because they are reporting the thought process, and their though process may change because they have to verbalize it;

- In retrospective think aloud, the session is recorded and when it is complete, users are asked to watch the recording and report on their thought process at the time. This technique does not have the risk of interfering in usability metrics, but it doubles the session length;

- In concurrent probing, the moderator will ask questions when the user does something interesting or abnormal, in order to understand the cause for that action. This technique is highly intrusive in the usability metrics, since it will interfere with the user thought process;

- In retrospective probing, users are questioned regarding interesting or abnormal actions and regarding their thought process after the session is over, in order to asses their opinion and understand any problems that occurred. While this method does not interfere with usability metrics, users may have difficulty remembering their reasoning during the session.

Since it was undesirable to interfere with the usability metrics and user time was limited, the moderation technique used for this thesis was retrospective probing, whose negative points were mitigated by the short projected session length.

Chapter 4

Implementation and result evaluation

The implementation followed a rapid iterative testing and evaluation (RITE) approach, where each component would be tested with one or two users for quick feedback and iterated upon until an intuitive and efficient solution was found. However, this meant that new and unforeseen problems would arise that would require changes in the framework that affected previously implemented components, either for technical reasons or to maintain consistency within the gesture vocabulary. This chapter will overview the framework from a technical perspective, explaining the architecture choices of the final result. Afterwards, there is an explanation of how the various individual user interaction components were implemented, followed by a description of the interface changes required to provide proper feedback to the user and ensure the accuracy is retained in the no-touch interface. It is then followed by a description of the process of adapting Glintts eResults to a no-touch interface using the implemented framework, along with the challenges that arose during this step. Finally, there is a section dedicated to the usability tests executed on the final prototype, describing how their process, results and retrieved conclusions.

4.1 Technical implementation

The Microsoft Kinect cannot natively interact with webpages, as the SDK is only available for C++ and C#. An initial approach for this problem consisted in a naive emulation of the mouse, where the native application would translate the hand movement into mouse movement at the level of the operating system, a grip and release would trigger a mouse press and mouse release event respectively and additional functionality could be added by emulating the scroll wheel and keyboard presses. This approach allowed users to press buttons without a problem, but had issues that caused it to be discarded:

- Handling two hands was out of the question. Since it was impossible to have two cursors on the screen, the user could not have feedback regarding the position of their second hand,

and neither could the webpage. Any event in the second hand would have to be dealt by the application.

- This approach added an unneeded layer of abstraction both for the native application and the webpage, since values other than cursor position could not be passed. This would later mean that, for example, changes in hand proximity to the screen would have to be abstracted to key presses.

- There were modularity limitations in this approach. This is a consequence of the lack of communication of values, but also because it meant a change in technology would result in a full re-implementation of the system.

It was clear that the Kinect data would have to be passed to the webpage and handled by it in order to optimize the feedback that would be served to the user and improve the accuracy of the data available to handle events. Following the study shown in chapter 3.3.4, it was decided that the communication would have to be implemented from scratch, using websockets.

4.1.1 Native application architecture

The native application makes use of the Kinect SDK 1.8 to retrieve the user data from the Microsoft Kinect and the Fleck library to set up websockets to which the framework connects in order to receive the user information. This application contains a `ConnectionManager` which is responsible for keeping track of all the connected clients and providing a static method for the rest of the framework to send messages. In addition, it is also ready to read any incoming messages from the clients, although, in the end, communication was kept unidirectional. In order to handle the Kinect, the aptly named `KinectHandler` is responsible for initializing the sensor and storing the information from the skeleton, depth and interaction data streams offered by the SDK, while also processing the skeleton info to determine the number of active hands and, in intermediate iterations of the framework, to process the hand positions when both hands were active. However, regardless of the fact that gesture detection makes use of skeleton data, gesture detection is not handled by the `KinectHandler`, but rather by the `GestureController`, because of the complexity of this operation which is best left to its own module, as explained the next section. The `GestureController` holds multiple instances of the `Gesture` class, which itself contains multiple segments, represented by the `IRelativeGestureSegment` interface, and a name, represented by the `GestureType` enumeration. On the other hand, the interaction stream, which contains data pertaining to the users hand state adjusted for interaction, is handled by the `InteractionHandler`, which sends both grip events and movement updates to the framework. When constructing the messages, the application makes use of the `Utilities` class, which was implemented to provide static utility methods that would be used throughout the whole application, but, in the end, was only used to properly convert `double` precision numbers to string bypassing regional decimal point settings, by enforcing a dot, which Javascript uses as the decimal separator, rather than a comma. The class diagram for this application can be found in the appendix A.

The Kinect initialization was another problem that had to be solved by the `KinectHandler`, because the interaction stream offered by the Kinect Developer Toolkit is normally reserved for WPF applications and because the Kinect itself may have hardware issues that should be solved at runtime. In order to solve the first of these two problems, a dummy `IInteractionClient`, which normally binds an interface component as interactive, was created to affect the entire screen, as suggested by András Velvárt [Vel13]. As for the hardware issues, these were mitigated by adding a listener to state changes in the Kinect sensor, which stops the streams from receiving data if the sensor is not ready, and re-initializes them if the sensor is ready. Nevertheless, irreversible hardware failures, such as the sensor shutting down due to insufficient power from the USB ports require reconnecting the Kinect and restarting the program.

4.1.2 Gesture recognition

In addition to the information offered by the Kinect SDK, of the hand position, pressure and grip, it was necessary to determine when the users waved their hands and when they made a pinch or swipe, in addition to the direction of the swipe or the pinch. In addition, it was desirable to make this recognition as expansible as possible, in order to accommodate future gestures.

It was decided that gesture recognition would be implemented in the native application rather than in Javascript because with the Microsoft Kinect, data from the elbow, shoulder and hip position was required to correctly identify these gestures and it was desirable to minimize the data sent and processed by the framework by sending only the hand data and the events. All the existing libraries for gesture recognition proved themselves outdated and, as a consequence, gesture recognition had to be implemented from scratch.

The implementation closely followed the one suggested by Vangos Pterneas [Pte14], where each gesture is specified as a sequence of gesture parts, where each of those parts acts as a state of a state machine, providing a common method that may return either a successful passage to the next part, a continuation of the current part, which is described as a pause in the gesture execution, or a return to the initial state, which represents a failure executing the gesture. Using this algorithm, the wave gesture was implemented as a repeated sequence of the hand being located to the left of its corresponding shoulder and then being located to the right of the same shoulder. On the other hand, an horizontal swipe is defined as a motion starting left of the left shoulder or right of the right shoulder, followed by the same hand between both shoulders and, finally, by having the hand beyond the shoulder opposite of the initial one. For this purpose, the left and right swipe have different parts, in order to make both the gesture validation and direction recognition easier. Lastly, the vertical swipe, which goes from top to bottom, is defined as a sequence where the hand starts above the shoulder, moves to a position between the shoulder and the hip, follows by going below the hip and finishes by returning to the position between the shoulder and the hip. This last step was added to avoid false positives, where the users would trigger a vertical swipe when disengaging or switching the hand they were using to interact. The pinch gesture, on the other hand, was not handled using this algorithm, because this gesture could be detected using only hand data and works more fluidly if the consequences of the pinch are triggered before the

gesture is complete, that is, if a zoom starts as soon as the user starts moving the hands away from each other and the zoom amount directly follows the distance between the hands.

4.1.3 Framework architecture

The framework follows a modular design, allowing the developers to change the implementation of how the interaction with a given component works, by swapping the values in `KinectOptions` module, which lists the possible modules in the comments.

The document, image and video modules offer a common interface that allows the developer to place one of these elements in a page by specifying the container div and the path to the file with the image, PDF or video content, by calling the function `init(container_div, source)`. When the module is no longer needed, the `clean()` function should be called to undo any changes performed on the container div when the module was initialized. However, the `KinectNavigation` module automatically handles this task, which means that if the developer uses the navigation module offered by the framework, the module cleanup process is opaque.

Despite the chosen modules, the `KinectConnection` module is always used to receive the JSON messages from the API, process the instruction and pass it on to the appropriate handlers. In addition, the kinect-framework.js file is also used as well, as it provides common utility that is necessary in all the states that the page may be, such as updating cursor position, pressure and grip, as well as triggering the appropriate events in the DOM elements annotated with classes relative to the framework. This file also initializes these elements accordingly and, as such, offers the `initializeKinectElements` function, that the developer should invoke when the page has loaded in order to allow the page to work via no-touch.

The framework also offers a lateral menu that is invisible until the user moves his or her hand to the right edge of the screen, and, to add buttons to this lateral menu, it is possible to register buttons in the `KinectSideMenu` module before initialization, so that they get properly initialized after being added to the DOM.

Finally, the framework also provides a stylesheet which defines the minimum dimensions for the elements interactive via no-touch and defines attributes for the elements that are necessarily used: the lateral menu and the cursors.

In conclusion, the framework offers a number of Javascript modules that can be swapped as the developer requires. The only interaction required by the developer to offer no-touch functionality to the application is to perform a general initialization when the page is loaded and to initialize the image, document and video modules as necessary. It should also be noted that the framework is completely compatible with Google Chrome, Firefox and Internet Explorer of version 10 and upwards.

4.1.4 API specification

The framework offers an API using websockets to communicate with the native application that gets the input data from the Microsoft Kinect. All the messages must be sent in JSON and must

Operation	Arguments
grip	hand_side, grip_event
hand_n_update	n_hands
movement	hand_side, x, y, press_extent, raw_press_extent
swipe	direction, time_taken
wave	none

Table 4.1: Operations supported by the API

contain a string labelled operation, indicating the type of information contained within the message. Table 4.1 summarizes the supported operations and their arguments.

The grip message represents a change in the hand state and, therefore, may represent either a grip or a grip release. The grip is an event, rather than a state passed along with the movement because the Kinect SDK provides the grip information in this manner. In these messages, the hand_side may be either left or right, depending on whether the information pertains to the right or left hand and the grip_event argument may be either grip or release, depending on whether the hand opened or closed the grip.

The hand_n_update message serves to inform the system of the number of active hands used. This message is needed so that the system can hide the cursor of the inactive hand or start displaying the cursor of the second hand, and to know when the user is disengaged from the system. The reason for this being handled by a message sent from a native application, rather than at the level of the framework itself is to ensure that the users hand goes beneath the hip before labelling it as inactive.

The movement message describes the current hand position and, as such, is the most frequently sent message, since, in order to offer smooth animation, the Kinect offers this information thirty times per second. Once again, the hand_side attribute represents the hand to which the message refers, while x and y are floating point numbers representing the position of the hand, ranging between zero and one, where zero represents the top-left corner of the screen and one represents the bottom-right corner. Both the press_extent and raw_press_extent attributes refer to the proximity of the hand to the screen, but, whereas the raw_press_extent is a direct measure of how much the arm is stretched, which is used by the framework for zoom operations using hand pressure, the hand_pressure is adjusted to make pressing a button smoother, by the means of calculations done by the Kinect SDK.

The swipe message describes, as the name suggests, a swipe by the user, which may be vertical or horizontal, as represented by the direction attribute, which may be left, right or top-bottom. This message is sent only when the motion has been completely registered by the native application and the time_taken attribute holds the time, in milliseconds, taken to perform the motion, which would be used to scroll in proportion to this time, if it had not proved itself an ineffective way to scroll.

Finally, the wave message represents a wave and, since it is used for engagement, it holds no arguments, because the only variables in a wave motion, which are the hand used and the time

taken to perform the wave, are irrelevant in the engagement process.

4.2 Component implementation and gesture vocabulary

In order to implement all the required components, a gesture vocabulary had to be designed for the whole system, in order to maintain consistency and to avoid overlaps between the various components. Table 4.2 summarizes the attempted components of the gesture vocabulary and the conclusions obtained from each one of them. An interesting conclusion from this table is that every gesture that was kept in the framework was of manipulative nature, where the user interacts with the element as if it were present in the air, with the exception of engagement, which made use of a symbolic gesture. The following sections details their design process, implementation and framework integration.

Context	**Hand state**	**Hand motion**	**System Response**	**Conclusions**
Images	One hand gripped	Omnidirectional movement	Image translation	Maintained in the framework, as it was intuitive and efficient
Images	Two hands gripped	Circular movement	Image rotation	Maintained in the framework, as it was intuitive and efficient
Images	Two hands gripped	Omnidirectional movement in the same direction in both hands	Image translation	Maintained in the framework, but generally less used than one handed translation
Images and documents	Two hands gripped	Hand movement in opposite directions, also known as a pinch	Zoom in or out	Maintained in the framework, as it was intuitive and efficient
Images and documents	One hand gripped	Move the hand towards or away from the screen	Zoom in or out of the image or document	Maintained in the framework as an alternative to the two handed approach
Scrollable elements	One hand gripped	Omnidirectional movement	Scroll equal to the movement ammount	Discarded because repeating the movement was tiresome

Scrollable elements	One hand gripped	Maintaining the hand at a certain position	Continuous scroll equal to the distance to the original grip position	Maintained in the framework, as it was intuitive and not tiresome
Scrollable elements	One hand gripped	Vertical movement of the non-gripped hand	Continuous vertical scroll equal to the vertical distance between the two hands	Discarded because it was asymmetrical and therefore unintuitive and could register false positives when the user intended to proceed to another gesture
Clickable elements	Hand grip followed by a release	None	Simulate a click	Maintained in the framework, as it was intuitive and efficient
Clickable elements	Open hand	Push towards the screen	Simulate a click	Maintained in the framework, as it was efficient and the user would become more proficient with it
Disengaged user	Irrelevant	Hand wave	System engagement	Maintained in the framework, as it caused few false positives
Videos	Hand gripped	Horizontal movement	Time seek within the video	Maintained in the framework, as it was efficient and intuitive
Videos	Hand open	Push towards the screen	Play or pause the video	Maintained in the framework, as it was efficient and the user would become more proficient with it
Any context	Irrelevant	Horizontal swipe	Navigate to the previous element in the page	Maintained as an alternative to button based navigation, as it was intuitive
Any context	Irrelevant	Vertical swipe	Navigate to the initial element in the page	Maintained as an alternative to button based navigation, as it was intuitive

Table 4.2: Summary of the attempted implementation of a gesture vocabulary

4.2.1 Engagement

In No-Touch Interfaces, the user is always tracked and, as such, may accidentally interact with the system when they have another intent. Because of this, there is a necessity for a gesture or signal that the user intends to interact with the system. Science fiction suggests that a wave by the user should initialize interaction and it is trivial to see the reason for this: a wave, as defined by the repeated sequence of moving the hand from left of the corresponding shoulder to the right, followed by a return to the left, is unlikely to be used in common conversation or actions. At the same time, waving is not tiresome if it is correctly recognized by the system, making it an intuitive choice for engagement implementation.

Another point that needs to be addressed is the disengagement, that is, when the user should be labelled as not interacting with the system. A naive approach would be to disengage the user when both of their hands are lowered beyond the interaction area or when the user is no longer found. This approach caused false positives when the user wished to switch hands and when the user attempted to perform a vertical swipe. This problem was mitigated with a timer, that would only trigger the disengagement if the user lowered their hands for three or more seconds, which eliminated false positives and had a minimal amount of false negatives.

An alternative approach to disengagement was considered to allow the user to rest their arms without disengaging the system. A use case for this would be during surgery, where the doctor may wish to interact with the system, interact with the patient and, later, interact with the system again without having to engage. While it seemed natural to use the same gesture for disengagement as the one for engagement, the wave gesture caused false positives during interaction or when the user would casually move their hand while browsing results. Due to the extensive gesture vocabulary, it was impossible to find another universal gesture that would not collide with other interactions and, as such, it is not recommended to use gestures to trigger a disengagement.

In summary, the implemented framework allows users to move their hands and see the hand projection replicating the movement, but they cannot press any buttons, nor are they highlighted, until the user waves to engage the system. This state is accompanied by a message indicated what the user should do, which fades out when the user engages the system and returns when the user disengages. To disengage, both the hands down and swipe approaches were implemented and the developer can alter the disengagement method in the `KinectOptions` module, although waving to disengage is not recommended.

4.2.2 Pressable elements

The interaction with pressable elements was designed with the goal of being analogous to a mouse click. This meant that there should be an action analogous to pressing the mouse down and another action analogous to a mouse release, which should result in a click if the user did not move the cursor outside the element. This analogy allows user to confirm whether their action was intentional and, since there is no fixed wait time, as found in hover based approaches, the users can learn how to use the system better and optimize their own capacity.

With this concept in mind, two complementary approaches were implemented: the press and the grip. The press works in the same manner as the one Microsoft uses and recommends, where the user can extends their arm towards the screen to trigger the mouse down portion of the click and contract it again to trigger the mouse up. The grip, on the other hand, was implemented because the two state nature of a grip and release could easily be compared to the mouse down and up states, respectively. In both these approaches, the cursor becomes anchored to the clicked element during the mouse down state and it is only released form the anchor if the user proceeds with the click or if his hand movement exceeds a threshold of 10% of the screen width. This anchoring was implemented to compensate for the jitter in the Kinect and involuntary movements by the user, thus increasing confidence that the element is still being pressed or gripped.

In conclusion, the framework offers the `kinect-pressable` class for any HTML element, which will adjust its dimensions to those necessary in order to achieve high accuracy, and trigger a click when the user presses it or grips and releases it, so that the developer can define the behaviour of the element by adding a listener to the click event, as they normally would in a WIMP interface. In addition, the element will also be the target of the `handenter` and `handleave` events, where the developer should add visual feedback that the hand is hovering over the element. The framework does not provide this behaviour by default, because the feedback may change from application to application, for design purposes.

4.2.3 Drop-down lists

What is commonly known in WIMP interfaces as drop-down lists could not work by dropping down in NUIs. This is already evident in touchscreen interfaces, such as in phones using the Android operating system, where drop-down lists cause a pop up with the available options. As such, the first step was to create this sort of pop up, which was achieved using the jQuery plugin colorbox, which allows inline HTML to be used inside a lightbox that darkens the screen outside the colorbox it generates and, as a consequence, maintains state changes between pop ups. However, since these drop-down lists could contain any number of elements, there was a need for a solution that could hold for any number of options of any size and merely listing the options would make them troublesome to navigate. To this effect, the framework calculates the number of options it can fit on the screen and creates a carousel, using the jQuery plugin Owl Carousel, where each page of the carousel contains as many options as possible. Nevertheless, navigation within the carousel was still problematic. The first implementation that attempted to solve this used an horizontal swipe to change the active carousel element. This implementation was unsuccessful, as it was tiring to repeatedly swipe in order navigate through a large amount of elements and moving a number of elements dependant on the swipe speed, where a faster swipe would move advance a larger number of elements than a slower one, caused precision issues, where it would be difficult for the user to move through a shorter number of elements. From this approach it could be concluded that it was necessary for a navigation that did not involve repeating gestures and, if the users accidentally skipped past their goal, they could easily undo the movement. With these conclusions in mind, two arrows, pointing left and right, were added bellow the listed elements, that

would advance forward or backward on the carousel one element every half second. On top of this, a zoom out filter was added, which created yet another carousel, where each button corresponded to a group of buttons of the previous carousel. This system propagated recursively until it could fit all the buttons in a single page, as seen in figure 4.1, where, after three layers, all elements were represented in a single page. Furthermore, a button was placed between the navigation arrows beneath the elements, that switched the displayed layer to the one that represents all elements and allowed the user to "zoom in" until they got to the layer with single elements.

In conclusion, the framework provides an adaptation of the HTML select tag, where the developer only needs to add the `kinect-combo` class to the element. When adapted, the default click behaviour is prevented, and replaced by a lightbox with a carousel where each carousel elements contains as many options as it can fit on the screen.

4.2.4 Scrolling

Scrolling, which is used for navigation within lists and documents, can be vertical or horizontal and may be used to handle elements whose total height or width is 50 times larger than the screen. An initial naive attempt was made where a simple grab and drag would move the scrollbar accordingly. However, this gesture proved itself tiresome in longer lists, since the user would have to repeatedly move their hand to one side of the screen, close the hand, move it to the other side and open the hand again. Users would become tired after performing this motion roughly three times vertically or five times horizontally. A conclusion retrieved from this attempt is that, although lists commonly alter between elements vertically and extend horizontally if the elements are too large, resulting in a scroll that is larger vertically than horizontally, in no-touch interfaces it is preferable to make the dominant scroll orientation horizontal.

The next attempt made use of both of the user's hands, where the user was be able to close one hand and the list would scroll an amount equal to the distance between the left and right hands. Therefore, to scroll vertically, they could, for example, close the right hand and leave it at shoulder height, while leaving the left hand at stomach height. The user could slow down the scroll by placing the left hand higher and, as a consequence, closer to the right hand. The scroll would stop when both hands were at a similar height or when the user would open their hand. Although this approach was less tiresome, it was less intuitive as a consequence of the asymmetry between the hands. Closing both hands to trigger the scroll mode was not an option to solve this, since it would increase the difficulty in understanding what hand was used as a reference point for the scroll difference.

Afterwards, an attempt was made where buttons with arrows would be place at the edges of the scrollable element and the user could hover their hand over these buttons to continuously scroll in the direction that the arrow was point towards. This approach proved itself both intuitive and not tiresome. However, it consumed some of the already scarce screen space and resulted in an increase of difficulty controlling the scroll in smaller amounts.

Finally, an approach was experimented that tried to match the strongest points of each of the previous attempts. This approach, like the drag to scroll approach, uses the grip as a clutch to

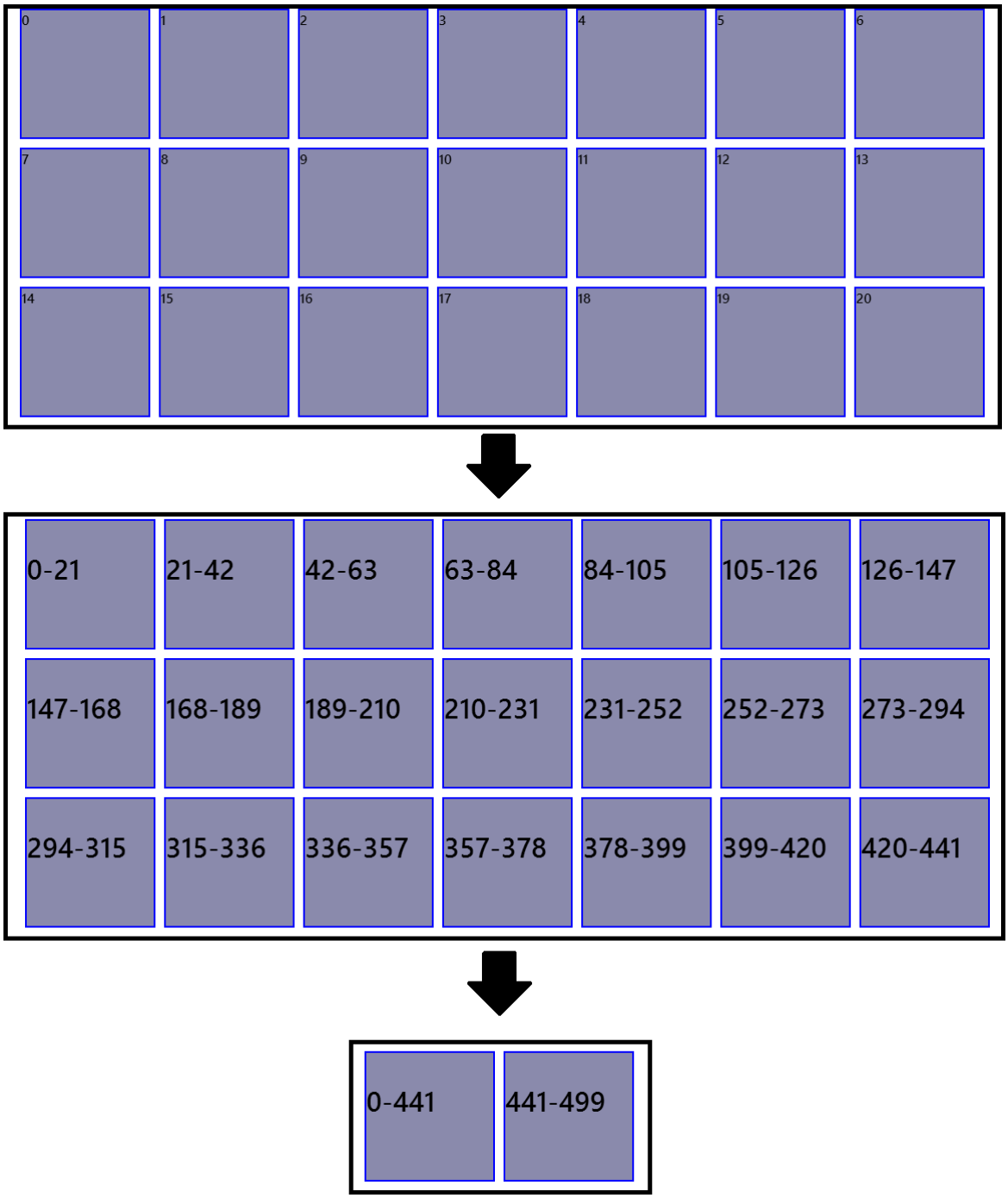

Figure 4.1: Example of the multi layered solution to carousel scrolling, in a situation with 500 elements where only 21 can be fit on the screen.

enable the interaction and sets it as the center point. If the hand is at a distance from the center larger than 10% of the screen width, the list or document will continuously scroll in the direction that the hand moved, in a speed directly proportional to the distance to the center point. This approach proved itself both intuitive and less tiresome than the others, although prolonged scrolls

upwards were still tiring after 30 seconds.

For the developer, the framework offers scrollable elements to the developer by offering the `kinect-scrollable` class, which, after initialization, offers the behaviour detailed in the last approach to the element. In contrast, scrolling in documents is offered by the document plugins and is independent of the `kinect-scrollable` class, despite offering the same behaviour.

4.2.5 Image manipulation

For this work, it was required that users would be able to perform basic image manipulation, in the form of translation, rotation and scaling transformations. Three modules were implemented to achieve this goal, one of them being button based and the other two gesture based, each with their own advantages and disadvantages.

The first of these approaches intends to emulate image manipulation in touchscreens, with a gripped hand replacing the fingers, where a pinch is used to zoom, a rotation of both hands or fingers is used to rotate and the movement of either a single hand or finger, or a movement of both in the same direction causes a translation. Implementing a single handed translation was trivial, as it only required a direct equivalence of the hand movement to a translation amount. However, implementing the two handed gestures required some further care and basic mathematics. For rotation, angle calculation requires the following steps:

1. When both hands are detected as gripped, the line slope between them is declined using the commonly known line slope formula: $m = \frac{Y_{left} - Y_{right}}{X_{left} - X_{right}}$

2. On every frame, the current line slope is calculated using the same formula.

3. The angle between the current slope and the original one is then found. It is known that the tangent of the angle between two lines is as follows: $\tan(\theta) = \frac{m_1 - m_2}{1 + m_1 m_2}$ Therefore, the angle between the slopes is the arctangent of the previous formula.

4. Since the previous formula calculates the acute angle between the lines, there is a need for adjustment if the angle is bigger than $\frac{\pi}{2}$ or smaller than $-\frac{\pi}{2}$. This situation is identified by a sudden shift in the signal of the calculated angle, either from a value higher than 1 to a negative value or from a value smaller than -1 to a positive value and it is fixed by subtracting the obtained angle from π in the former situation or by adding it to $-\pi$ in the later case.

5. Finally, the previous angle is subtracted from the current one to determine the angle to be added to the image.

As for the implementation of a pinch for the zoom, this required little more than a simple difference in hand distance between the previous frame and the current one. Since a scale of 1 corresponds to the image at the original size, 0.1 to the image ten times smaller and a scale of 2 is an image twice as large, and a difference of 50 pixels is a very small one, the distance difference is divided by a

constant to fit the difference to the scale. This constant was set to 1000 and performed adequately. Finally, if the scale is negative, which occurs if the hands are approaching each other and therefore have a decreasing distance, it is adapted to a value between 0 and 1, to represent a zoom out, by dividing 1 by the absolute value of the scale. The two handed translation, which occurs when the user with both gripped hands moves them in the same direction, was implemented by verifying if the difference in both horizontal and vertical position has the same signal for both hands, that is, if both move in the same direction. If this type of translation is detected in a given frame, zoom is disabled for that frame, since the user may not be moving their hands at the same speed, which could trigger an accidental zoom. The pseudocode for this entire interaction can be found in the appendix B.

The second approach consisted in an attempt to make full use of the three dimensional interaction offered by no-touch interfaces in order to allow the user to control the system with a single hand. For this approach, the translation was, once again, executed by grabbing the image and moving it in the 2D space, while the zoom could be executed by grabbing the image and moving the hand towards the screen and away from it. However, this approach was problematic when dealing with rotation, due to the fact that both the 2D and 3D space were exhausted, rotation could not be implemented without adding a second hand for control.

Finally, an approach with buttons that would change the operating mode was attempted. This approach added three buttons to the bottom of the screen, which the user can press to alter between translation, zoom and rotation. Figure 4.2 shows these controls, which are placed on the bottom of the screen, in order to minimize the effort required to access them. In this figure, no mode is selected, as the selected operation has a different background color.

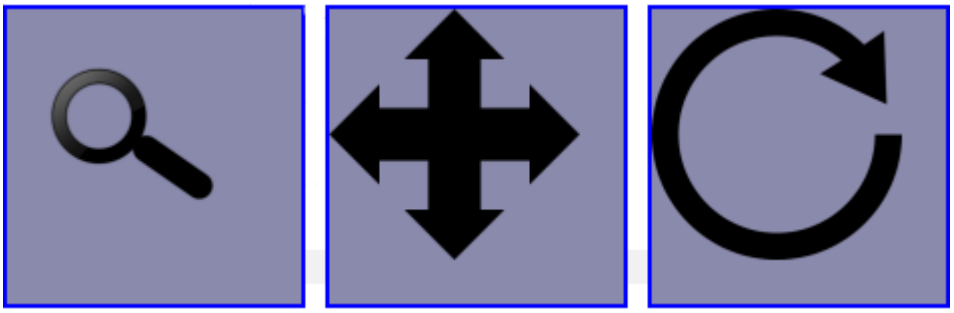

Figure 4.2: Controls used to switch between operations when manipulating an image.

In this implementation, when the translation or zoom modes were active, they would behave in the same way as the one described in the previous approach. In addition, rotation is handled with an algorithm similar to the one used for rotation with two hands, using the center of the image as the reference point to calculate line slopes, along with the hand position.

From a technical point of view, the jquery.transform.js plugin was used to apply the CSS transformations to the image with cross-browser compatibility. Although this plugin offers the option to either directly transform the image or to animate it to the desired transformation, a direct

transform was used because the Kinect's framerate was enough to simulate animation by simply changing the image's properties.

Concluding, the framework provides two image manipulation modules, which can be altered in the options module, and allow the developer to add an image ready to be used via a no-touch interface. One of these modules works like touchscreens and requires both hands to be used in entirety, another is one handed but is requires extra screen space and extra time to switch between operations.

4.2.6 Document manipulation

Since the gesture vocabulary for scrolling and image elements was already defined, finding a gesture vocabulary for documents was simple, since the actions are equal and consistency in the gesture vocabulary of the system is desired.

However, from a technical point of view, implementing document manipulation was non-trivial. When a web browser finds a DOM element of type object containing a PDF, it calls its own reader plugin, either native to the browser or installed by an user. This means that the actual characteristics of the PDF object are opaque to Javascript and CSS and, which means that, by default, handling scroll and zoom programmatically is impossible. Three alternatives stood out to solve this issue:

1. Adobe provides a Javascript API for Adobe Acrobat [Inc07], which would mean that the PDF would only be usable if the browser used Adobe's reader as a browser extension.

2. Google provides an embeddable PDF viewer [Goo09], with its own Javascript API, that uses Google Drive as the backend engine to render the PDF.

3. Mozilla authored an open-source reader, pdf.js, which is controllable via Javascript [Moz14]. This reader can be used as a browser extension, like Adobe's reader, or as a page hosted either remotely by Mozilla, or locally, in whichever server the developer wishes.

Since it was desirable to make the system functional on any machine configuration and publicly linking a document was out of the question for a matter of patient privacy, the chosen option was to use pdf.js locally. Although pdf.js provides the tools to fully customize the reader's functionalities and aspect, the default viewer was used with the toolbar hidden, by embedding the viewer.html file in an iframe.

The implemented gesture vocabulary, with the last scroll implementation described in chapter 4.2.4 and the zoom approaches described in the chapter 4.2.5, performed adequately. Nevertheless, zoom took more time to get accustomed to in PDF files than in images, because the system response time was slower rendering a PDF at a higher or lower zoom than applying a CSS transformation to a DOM element.

Similarly to the structure in the image modules, the document modules also allow the developer to add a PDF controllable via no-touch and can be switched by changing a single line in the `KinectOptions` module. However, developers should be aware that, unlike images, pdf.js loads

files via a `XMLHttpRequest`, which means that this module will not work unless the page is accessed via HTTP or HTTPS.

4.2.7 Video manipulation

Since the actions in video manipulation are limited to a binary state change, between pause and playing, and manipulation of a value within a range, the gesture vocabulary for this component can be mapped in the 2D space and therefore be easily adapted from the interfaces found in touchscreens. Furthermore, from a technical standpoint, HTML5 makes embedding videos that are controllable via Javascript trivial, with the only drawback being that the toolbar and frame are browser dependant and not configurable.

With this in mind, an approach similar to the video controls commonly found in touchscreens was followed, where a click, either in the form of a press or grip and release, switch the video state between paused and playing, and holding and dragging right or left seeks forward or backward within the video, where the entire video acts as the area that can be dragged, rather than the small progress bar.

4.2.8 Navigation

Navigation is necessary in any interface in order to avoid dead ends that force a restart of the system or page and to allow the user to easily undo mistakes. For this work, it was desirable to be able to go back to the last screen and to return to the home screen and there were two approaches to this problem, both based on already existent mechanisms in touchscreen interfaces.

The first approach is by the simple use of buttons with appropriate iconography, with an arrow pointing to the left symbolizing the return to a previous element and a house icon would return to the first page of the application. However, since screen space is a commodity in no-touch interface, these buttons were inserted in a lateral menu, that slides in view when the user's hand collides with a hitbox that occupies roughly 5% of the screen's width and is positioned at the right edge of the screen. This menu proved itself useful beyond navigation, since it could also act as a toolbar for content manipulation, such as changing the type of list view or filtering results. Figure 4.3 shows this menu when only the navigation buttons are present, in opposition to figure 4.4, where the navigation buttons are coupled with buttons to interact with the displayed content. As such, this approach was efficient, but costly in terms of screen space, as the menu space could be useful for the contextual actions.

The other approach was based on the concept of swiping to dismiss an element, such as the one present in tablets running Windows 8 [Lin12]. In this approach, a horizontal swipe would return to the previous element and a vertical swipe would return to the first page. Both of these gestures caused false positives that were successfully mitigated. First, the horizontal swipe would trigger a second time when the user returned their hand to the initial state after an intentional swipe. This was mitigated by adding a cooldown period, to swipe navigation of five seconds, as suggested by Chen in a recent article [Che14], which proved enough to allow the user to return their hand

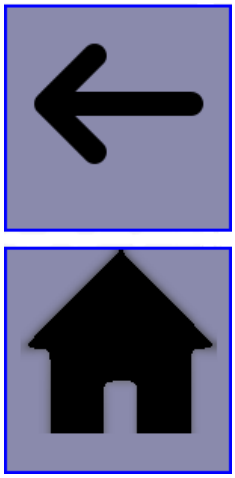

Figure 4.3: Navigation buttons in the lateral menu. These appear on the right edge of the screen when the user moves their hand to the edge of the screen.

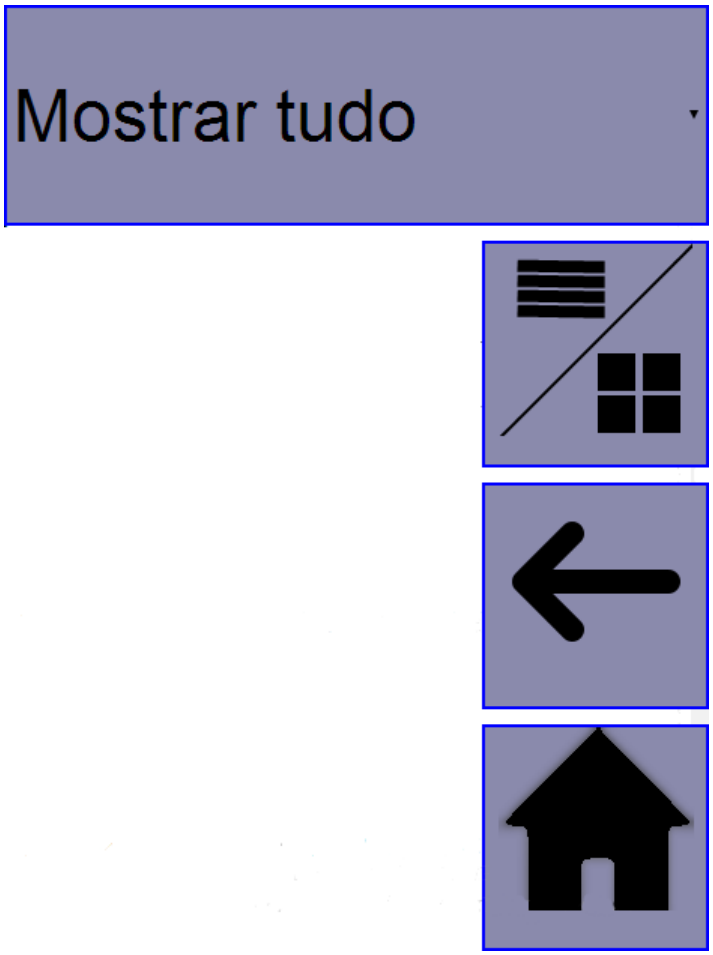

Figure 4.4: Lateral menu with the navigation buttons and contextual manipulation buttons.

to the natural hand position. On the other hand, the vertical swipe would cause false positives when the user intended to switch hands or disengage the system without returning to the start. This false positive was minimized by adding an extra step to the definition of a vertical swipe: rather than define a vertical swipe as a movement where the hand starts above shoulder height and keeps moving until it is at hip height, an addition was made where, after the previous definition, it should also return to a position above the hip. However, even with these adjustments, swipe based navigation caused more false positives than menu based navigation.

In conclusion, the framework implemented two methods of navigation, in the form of a spatially expensive menu navigation and a spatially free but error prone swipe navigation. The developer can switch between these options in the`KinectOptions` module and the navigation method will be applied to the entire application, since switching between navigation mode would cause a lack of consistency and therefore would confuse the user. Additionally, the developer may also want to perform a particular action when the user navigates, such as changing information in the page's header. To solve this, the observer design pattern was used, as it is possible to register handlers, that take no arguments, and are called both when the user returns back and when the user returns to the home page.

4.3 Feedback and display

In addition to the gesture vocabulary necessary to control no-touch interfaces, it was also necessary to determine rules for the information display. This meant defining minimum dimensions for an element to be interactive at a high accuracy, determining how to give feedback of the current state of the users hands, which provided extra challenges when the users needed both hands, due to limitations of the Kinect SDK and defining how elements should change their appearance in response to user actions.

4.3.1 Element dimensions

Due to the loss of precision in no-touch interfaces, minimum dimensions must be set in order to allow users to uniquely select and interact with elements without error. Microsoft's recommendations, shown in table 3.1 in chapter 3, define these dimensions as ranging between one eighth and one ninth of the screen's width and were used as the basis for the framework's minimum dimensions for pressable elements.

Since the dimensions recommended by Microsoft occupy such a large percentage of the usable area in the screen, attempts were made to decrease the size of the buttons while maintaining accuracy. For this purpose, four buttons were placed in the screen, with a margin of 10 pixels between them and a random order would be generated to click on those buttons. The first experiment used buttons 10 pixels smaller than the recommended dimension suggested by Microsoft for 720p screens, and, if the accuracy and user confidence were maintained, the next iterations would use smaller dimensions, until an unacceptable size was found. However, in this first iteration, buttons were already noticeably harder to grip, even for a single user that had been frequently using the

Kinect for various months. Later, upon the implementation of lists, where elements were listed vertically and each of them occupied the full width of the page, a new experiment was executed where, similarly to the previous one, smaller heights would be tested until there was a drop in user accuracy or confidence. In this case, where the elements occupied the full width of the page, heights 30 pixels smaller than the ones recommended by Microsoft still proved efficient on a 720p screen. This leads to the conclusion that the minimum width or height of the elements can be decreased if there is an increase in the total area of the element.

To implement these dimensions, the framework offers a stylesheet that contains responsive dimensions for the `kinect-pressable` class, following the dimensions suggested by Microsoft, and dimensions for elements containing both the `kinect-pressable` and `kinect-list-element` classes, with 20% smaller minimum height.

4.3.2 User cursor

The most basic building element of the interface was to represent what the system was seeing, which meant displaying the position of each hand and its state. To achieve this, a solution similar to Microsoft's UI components was followed: there is a transparent cursor following the hand movements, whose interior is filled with a colour as the hand approaches the screen, being completely full when the user is performing a press action and there is a cursor distinction between an open and a closed hand, as shown in figure 4.5. Regarding the dimensions of the cursor, they should be equal to the minimum dimensions of any interactive component, in their smallest state, in order to be able to easily interact with them while being able to uniquely identify the element that is being interacted with. To achieve this dynamic dimension, 720p and 1080p versions of the cursor were created and the system fetchess them accordingly, depending on the screen resolution. Nevertheless, these are not comprehensive of all possible resolutions and a dynamic approach that would resize the cursor SVG using Javascript would be preferred but such work fell out of the scope of this thesis.

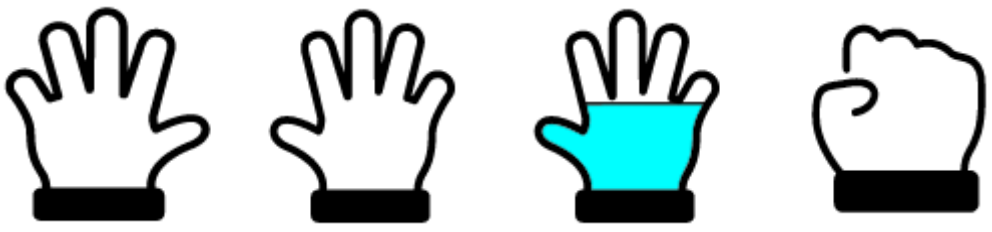

Figure 4.5: Cursor for an open left and right hands, followed by a right hand cursor with the arm extended at 50% of the necessary amount for a press and by a closed hand cursor.

To handle the cursor filling, multiple images were created, with different fill amounts, in increments of 10% and, when the framework received the pressure amount from the native application, it rounds this pressure to the first decimal point and, if it is different from the result in the previous frame, it hides the cursor with the previous pressure and displays the new one. Although a more

precise fill would have been achieved by dynamically filling the SVG using Javascript, the feedback provided by switching the displayed image proved itself good enough that it did not justify the extra effort that would arise from implementing the dynamic fill.

Concluding, the framework implements a small tree-like structure, where each hand has a total of twenty possible states, depending on its pressure and grip state, with only one of them being visible, as can be seen in code extract 4.1, which exhibits a sample of this structure, in a situation where only the right hand is visible and is at the minimum possible pressure.

Code extract 4.1: Structure of the DOM containing the cursor data

```
<div id="right" class="follow">
        <div id="rightopen">
                <object id="rightsvg_0" data="hands_720/
                    cursor_open_0.svg" type="image/svg+xml"></
                    object>
                <object id="rightsvg_10" data="hands_720/
                    cursor_open_10.svg" style="display:none" type
                    ="image/svg+xml"></object>
                ...
                <object id="rightsvg_100" data="hands_720/
                    cursor_open_100.svg" style="display:none"
                    type="image/svg+xml"></object>
        </div>
        <div id="rightclosed" style="display:none">
                <object id="rightsvgclosed_0" data="hands_720/
                    cursor_closed_0.svg" type="image/svg+xml"></
                    object>
                <object id="rightsvgclosed_10" data="hands_720/
                    cursor_closed_10.svg" style="display:none"
                    type="image/svg+xml"></object>
                ...
                <object id="rightsvgclosed_100" data="hands_720/
                    cursor_closed_100.svg" style="display:none"
                    type="image/svg+xml"></object>
        </div>
</div>
<div id="left" class="follow" style="display:none">
        <div id="leftopen">
                <object id="leftsvg_0" data="hands_720/
                    cursor_open_left_0.svg" type="image/svg+xml">
                    </object>
```

```
                <object id="leftsvg_10" data="hands_720/
                    cursor_open_left_10.svg" style="display:none"
                     type="image/svg+xml"></object>
                ...
                <object id="leftsvg_100" data="hands_720/
                    cursor_open_left_100.svg" style="display:none
                    " type="image/svg+xml"></object>
        </div>
        <div id="leftclosed" style="display:none">
                <object id="leftsvgclosed_0" data="hands_720/
                    cursor_closed_left_0.svg" type="image/svg+xml
                    "></object>
                <object id="leftsvgclosed_10" data="hands_720/
                    cursor_closed_left_10.svg" style="display:
                    none" type="image/svg+xml"></object>
                ...
                <object id="leftsvgclosed_100" data="hands_720/
                    cursor_closed_left_100.svg" style="display:
                    none" type="image/svg+xml"></object>
        </div>
</div>
```

4.3.3 Two handed interaction

As described in 3.3.3, each hand has its own physical interaction zone and, therefore, its own coordinate system. Since this system was designed for one handed interactions, this caused problems for two handed interactions. Figure 4.6 is an example of a problem caused by not adapting the hand coordinates: the users would see their hands in positions that did not correspond to reality, such as seeing their right hand on the left edge of the screen and the left one on the right edge when they were approaching each other.

The initial solution to this problem consisted in using a common coordinate system. To achieve this goal, skeletal data, rather than interaction data, would be used when the users raised a second hand. In order to create these coordinates, a center point would be created in the X-axis, in the user's hip center, and the edges of these coordinates would be located at the right of the right shoulder and left of the left shoulder, in a distance equal to the distance between a shoulder and the midpoint. This approach failed for two reasons:

- As figure 4.7 illustrates, the coordinate change when the second hand was raised or lowered was not intuitive, as the other hand would jump from its position in the physical interaction zone to the one in the two handed coordinate system or vice-versa. This was particularly

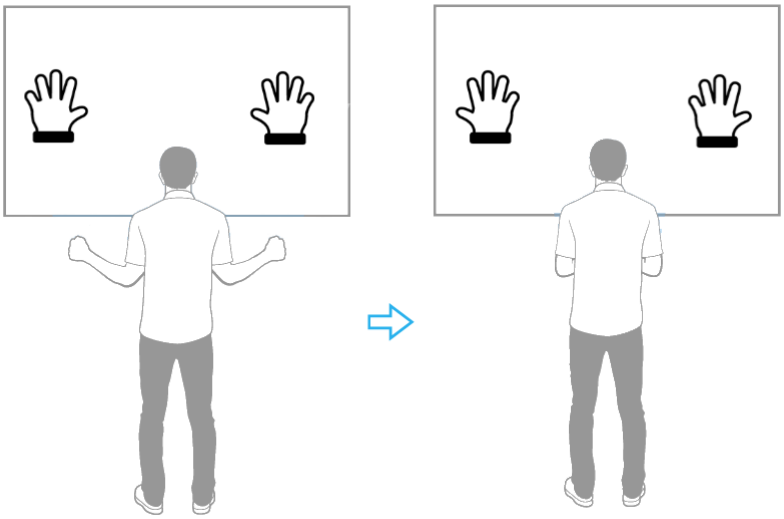

Figure 4.6: Example of a problem caused by not adapting the hand coordinates

problematic if the users slightly raised their hand without the intent of using it for interaction.

- These coordinates assumed the hands could leave their own physical interaction zone without further consequences. Unfortunately, as soon as the left hand started approaching the right side of the torso or the right hand approached the left side, the grip tracking was lost and the hand was assumed as open. This resulted in erratic switched between an open and closed hand which tended to break the interaction.

The following approach tried to continue using the interaction stream, while adapting the physical interaction zone. Since the API does not allow editing of the boundaries of the physical interaction zone, an attempt to stretch it was made, by multiplying the x-position by a constant, such as two, if the position was smaller than the mid-point of the physical interaction zone (0.5), and dividing by the same constant if the position was higher. However, since the values frequently escape the boundaries between 0 and 1, the results of this operation worked differently if the hand was further to the left, and therefore the position value was negative, than if it was further to the right, which meant a position value higher than 1. In practice, this translated into uneven movement speeds between a movement to the left and a movement to the right and was swiftly discarded.

Finally, a successful approach was elaborated based on the assumption that there is no interaction where the user must cross his arms. This was a safe assumption to make, not only because crossing the arms felt unintuitive, but because the Kinect itself was prone to jitter and errors where the hand would be assigned to the wrong arm, if the arms were crossed. With this assumption in mind, it was possible to enforce this restriction to the cursor representations and, therefore give the user feedback that his hands are as close to each other as the Kinect can handle. Figure 4.8

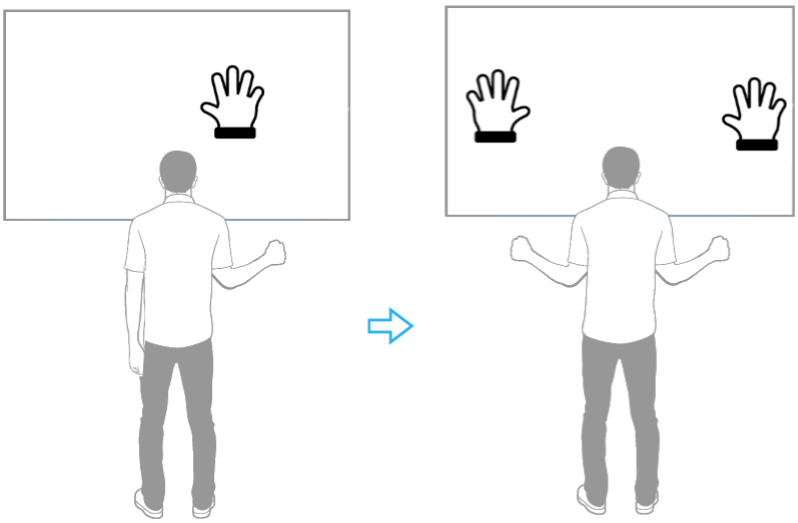

Figure 4.7: Example of a problem caused by switching the coordinate system on the fly. The suddenness of the movement is what makes this situation troubling.

shows this behaviour: although the Kinect provides the same coordinates as the ones in figure 4.6, but the framework assumes stops each hand cursor from moving beyond the other. This approach allows the user to move their hands as fluidly as if they were using a single hand, with no sudden changes to the hand position and with realistic feedback of their hand positions.

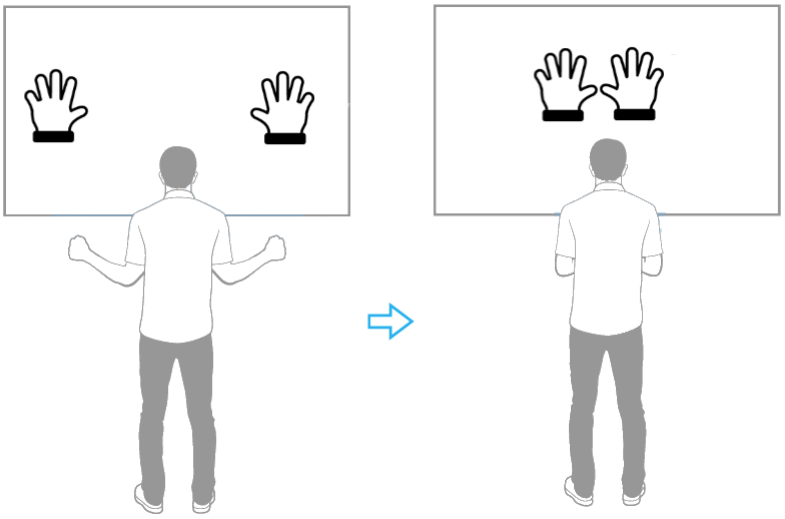

Figure 4.8: Intended behaviour for two handed interaction, which was implemented in the framework by forbidding crossings.

Another problem that arose regarding two handed interaction would be when to trigger it, that is, when to show or hide the cursor of the second hand and put into effect the measures that

attempt to fix the problems caused by the physical interaction zone. This was implemented by merely detecting whether the hand is above the hip, although there were attempts that added the verification of whether the hand was closer to the screen than the elbow, but, nevertheless, this addition caused too many false negatives.

4.3.4 Interaction feedback

In order to provide proper feedback, it was necessary to provide feedback to the user actions beyond the changes in user cursor position and image. More specifically, it was necessary to provide feedback about the state of the elements on the screen and whether the system considered that they were being interacted with. As such, the implemented components offer the following feedback:

- Pressable elements are highlighted when the users hover their hand over them. As previously mentioned, to accomplish, a `handenter` event is triggered on a `kinect-pressable` element when the hand is over it and a `handleave` event is triggered when it leaves. The handler for these events is up to the developer, because they may wish to toggle the highlight with a different colour for design purposes. Nevertheless, it is recommended to use a more vivid colour than the default one for the element, in order to make the highlight effective, similar to what was implemented in the eResults prototype. In addition, when the user presses the element, the element is made 15% smaller to increase user confidence, in an analogy to how a gripped rubber would be compressed by the grip or how a pressed button is moved further away and therefore perceived as smaller. This feedback is inspired by the one offered by the Microsoft controls for WPF applications, which also change dimensions to signal that the user is pressing an element.

- Scrollable elements feature the default scrollbars provided by the web browsers in order to leave the user aware of the remaining elements, as found in both WIMP interfaces and in other NUIs.

- Images act as their own feedback, by their changes in rotation, translation and scale. Nevertheless, the image is also confined to a container, which gains scrollbars if part of the image is hidden beyond the container bounds, as show in figure 4.9.

- Documents also act as their own feedback for the zoom operation and, since they are scrollable elements, they also make use of scrollbars.

- Videos act as pressable elements for the pause and play operations and, therefore, also become smaller when the user is pressing or gripping it. In addition, videos also feature the default progress bar native to HTML5 videos, allowing the users to locate their current location within the video time frame.

In addition to the feedback offered by each of the individual components when they are on the screen and the cursors, there is also engagement feedback, in the form of a rectangle with text

Figure 4.9: Example of a image larger than its container, with scrollbars indicating the amount of unseen content.

located at an absolute screen position, indicating the user how to engage, as shown in figure 4.10 that slowly fades out when the user engages the system and fades in when the user disengages the system.

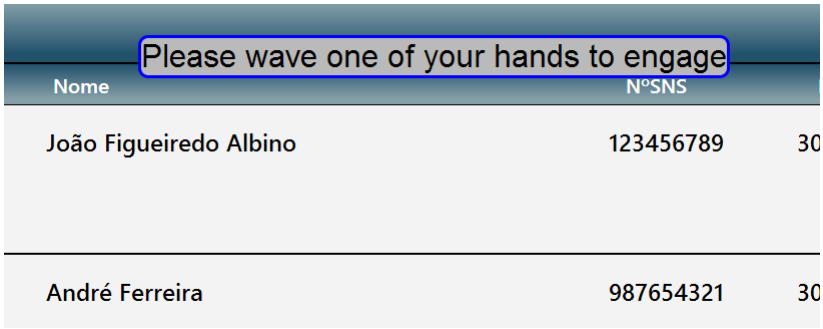

Figure 4.10: Engagement text in the developed prototype.

4.4 Building the eResults prototype

After the individual components were implemented, adapting a prototype of the eResults system would provide the proof that the framework correctly eased the adaptation of existing interfaces, while also providing a usability testing ground for the framework.

The first screen that was implemented was the patient selection, out of a list of patients. As seen in figure 4.11, this consisted of a simple list where each element was pressable and would replace the patient list by a results list, and a header common to the entire web application, whose

information would change depending on the selected patient. This adaptation required the following input beyond creating the markup listing the elements and header, as well as stylizing the colors:

- The `kinect-pressable` class was added to each list element;
- Listeners for the `handenter` and `handleave` events were added to the list elements, adding and removing the `hovered` class respectively;
- A listener was implemented for the `click` event, hiding the patient list and showing the results, while also pushing the patient list to the navigation stack;
- A call was added to the `initializeKinectElements` function offered by the framework.

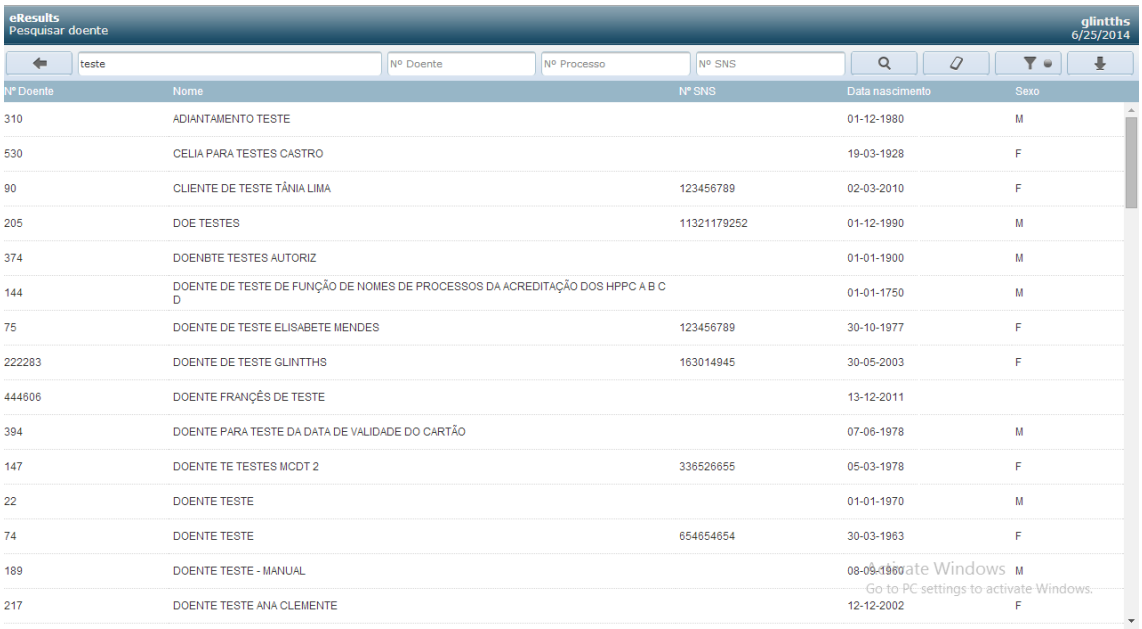

Figure 4.11: Patient listing in the eResults application.

These steps managed to give the correct functionality to the page, as well as the dimensions required to interact with it. However, a previously unnoticed problem arose, which was the necessity to make the text readable at a distance, while making sure that the text was small enough to fit in the screen. For this purpose, a font was needed with the largest x-height possible, which is a critical factor for readability [Car10]. Nevertheless, Arial, a font featuring an x-height of 0.518, proved itself too large, as text from one row in a column began to overlap into the following column. The font that was ultimately chosen was Segoe UI Semibold, which features an x-height of 0.500 and allowed the text to fit inside their respective columns at a font-size of 24px in a 720p screen. Figure 4.12 shows the implemented patient listing, with the correct dimensions. While it may seem that the font size could be increased even more to increase readability, it should be noted

that not only can the patient name and number occupy much more space, but the results listing would prove itself much more text intensive and therefore the typography should accommodate such environment.

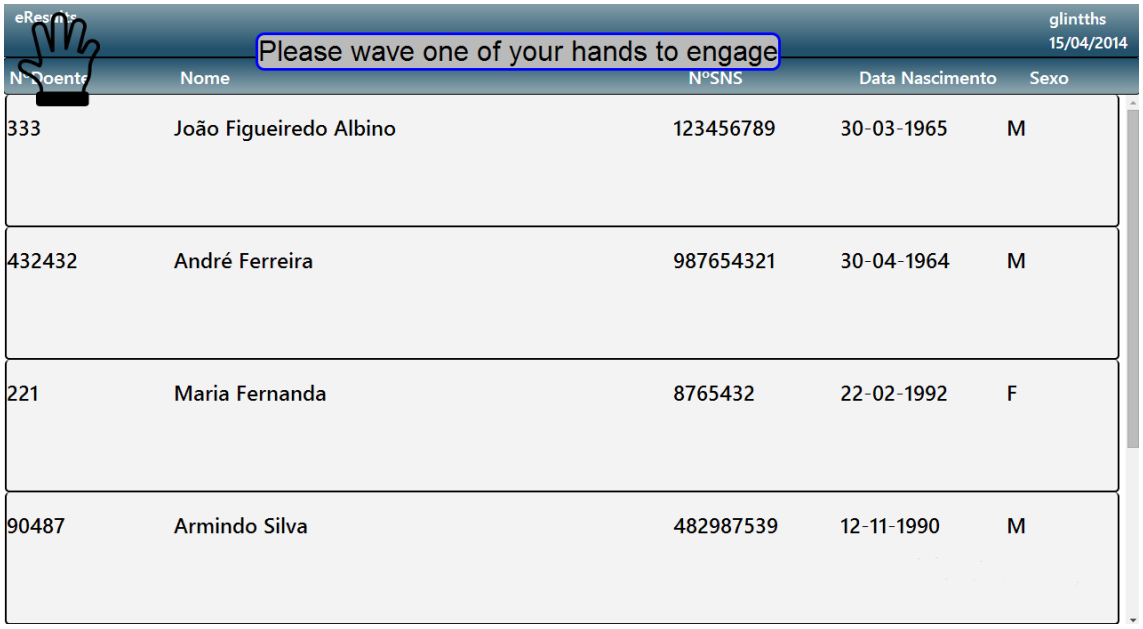

Figure 4.12: Patient listing in the implemented prototype.

Afterwards, the next step would be to implement the patient results listing and the switch to the image, document or video that they specify. Since the work done for the patients list already took care of adding the initialization and the listeners for the `handenter` and `handleave` events, all that was left was, once again, to add the `kinect-pressable` to the elements of this new list and set the click handlers to initialize either the image, document or video module with the popper content, which corresponds to a single function call. Since the framework already handled the interaction with these results and the navigation within the page, all that was left was to add buttons to interact with the results listing, as a proof of concept. For this purpose, the results listing would be able to be filtered by result type and be able to change the view, from a vertical list, to a grid list. In order to achieve this, a button and a drop-down list were created and added to the lateral menu, by merely creating them with the appropriate click handler to change the view on the button and the handler for the drop-down list to hide the filtered elements upon an `onchange` event, and then add them to the `KinectSideMenu.menu_elems` array before calling the `InitializeKinectElements` function.

4.5 Result validation

In order to validate the implemented framework from a usability standpoint, usability tests were carried out on the eResults prototype, with different gesture vocabularies, with the aim of gathering both objective and subjective data from users with little to no experience with the Microsoft Kinect. The following sections describe how the tests were carried out and the results from these tests.

4.5.1 Test methodology

In order to validate the results, the prototype would be experimented with different module interaction variations, in order to assess the efficiency of each interaction method. The tests were executed following a retrospective probing methodology, where the user would be asked questions about their thoughts and actions after the tasks were complete, in order to avoid interfering with the usability metrics. Additionally, the browser window was recorded during the session, to posteriorly count the number of errors and accurately measure the time taken to complete the tasks.

The users would use one of two sets of interaction methods, where one group would press buttons by extending their arm, interact with images and documents in an analogous manner to smartphones and navigate within the page using buttons from the lateral menu, while the other group would press buttons by gripping and releasing the button, interact with images via buttons that switch between translation, rotation and zoom, zoom on documents by moving the gripped hand towards or away from the screen and navigate through swipes. The group of users that made of use of the former set of gestures was labelled group A, while group B used the later one. It was desirable that each of these groups was tested with at least five users, since, according to Jacob Nielsen, this would be enough to find 85% of the usability problems, although ideally the system would be tested with fifteen users, in order to find all the problems.

The objective metrics for efficiency were the error count for the tests and the time of execution, with an error being defined as either a false positive or false negative when attempting to execute a gesture, or a movement having a result more or less intense than the user intended in document, image and manipulation. The subjective metrics were intuitiveness and tiredness in a five point Linkert scale, where a one would mean less intuitive or more tiresome, while a five reflected a more intuitive and less tiresome experience. Both the objective and subjective metrics were collected for both the entire application and each individual component: engagement, button pressing, navigation, image manipulation, document manipulation and video manipulation. In addition, the background metrics for the users were the previous experience with the Kinect, since this experience could make it easier for the user to get accustomed to the interface and the dominant hand, because the lateral menu was placed on the right of the screen, making the interface asymmetrical, and because Microsoft recognizes that grip accuracy is worse for the left hand than the right hand [Mic13a].

The tests were run on a Samsung ATIV Book 4 laptop running Windows 7, featuring an Intel Core i5-3230M processor, 8GB of RAM and an AMD Radeon HD8750M graphics card connected

to a Samsung monitor measuring 24 inches and featuring a resolution of 1920 pixels of width by 1080 pixels of height, along with a refresh rate of 60 Hz . It should be noted that this machine was different from the one used in development, a Lenovo R500 with an Intel Core 2 Duo T6670, 4 GB of RAM and an integrated graphics card running Windows 8.1, because this machine had noticeable lag when the screen was recorded while prototype was used, which would severely interfere in the usability metrics.

As for the physical conditions of the tests, they were executed with the Kinect sensor below the monitor, as seen in figure 4.13 and with the users standing at a distance of three meters from the monitor, with a table of sixty centimetres of height between them.

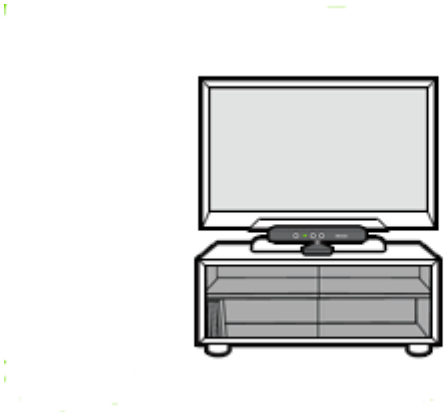

Figure 4.13: Diagram of the Kinect placement during the usability tests.

The sessions were planned to last ten minutes, with the following breakdown:

1. Upon arrival, users would be introduced to the objective of the program and the test and their background information would be gathered. This step would take one minute;

2. Afterwards, the users would be demonstrated the controls they would use, by showing the engagement process, patient selection, navigation and image, document and video manipulation. This step would take two minutes;

3. The page would then be refreshed, screen recording would start and the user would be given the following instructions, which were selected to take up to five minutes:

   (a) Engage with the system and select the second patient in the list, named "André Ferreira".

   (b) Select the document called "Relatório de MCDTs";

   (c) Zoom out until the entire document can be seen, zoom in on the patient name, which was located on the top of the page and, finally, scroll to the signature of the doctor, which was located in the bottom half of the page;

   (d) Return to the patient listing;

(e) Select the patient named "Pedro Azevedo", which was the last element of the list and would therefore require scrolling down the list;

(f) Select the image with the name "Ecografia 2ª opinião", which would also require scrolling;

(g) Flip the image vertically and zoom in to make the text "Hernia Inguinal" occupy the full height of the content area;

(h) Return to the results selection and select the video "Sessão de ultrasom" which was the last element in the list and would, again, require scrolling;

(i) Start the video;

(j) After fifteen seconds, pause the video and move to the five minutes mark in the video, with an error margin of twenty seconds;

(k) Finally, play the video and after fifteen seconds, pause the video again.

4. Finally, the users would rate the intuitiveness and tiredness of each of the components and the system as a whole and offer any opinion or insight they might wish. In addition, if there was any particular anomaly during the session, it would be discussed during this step. The projected time for this step was of two minutes.

4.5.2 Results

The total number of users participating in the usability tests was ten. None of the users had previous experience using the Kinect and one of them was left handed, who was assigned to group A.

Table 4.3 shows the results for the components with interaction sets common to both group A and group B: engagement and video manipulation. It can be noted that the video component performed poorly and was not well received by the users. The reason for this was that the progress bar worked as poor feedback, by not updating properly when the users tried to seek within the video was too small to be properly seen at a distance of three meters and it would not update smoothly along the hand movement. The first of these problems would not be relevant in Internet Explorer 11, since it features a much larger player interface for videos, as shown in figure 4.14, and the second one may not be problematic with a video using different encoding. Nevertheless, the ideal solution for this problem would be the development of a video player optimized for no-touch interfaces, that would perform equally in all browsers, but this would fall out of the scope of this work. As for the engagement, all the errors were false negatives, as many users were unable to perform the wave at their first attempt, but would easily correct their motion and be able to use the system in an average of six seconds, which they found an acceptable time to begin using the application.

Table 4.4 exposes the means of the results obtained in group A. With the exception of the video component, the interaction set for this group worked remarkably, with the users finding the image manipulation and navigation to be flawless. On the other hand, there was an issue with one

Component	Time taken (minutes)	Error count	Intuitiveness rating	Tiredness rating
Engagement	0:06	0.9	4.6	4.6
Video manipulation	1:31	8	2.7	4.2

Table 4.3: Table with the results of the usability tests for the elements common for both group A and group B.

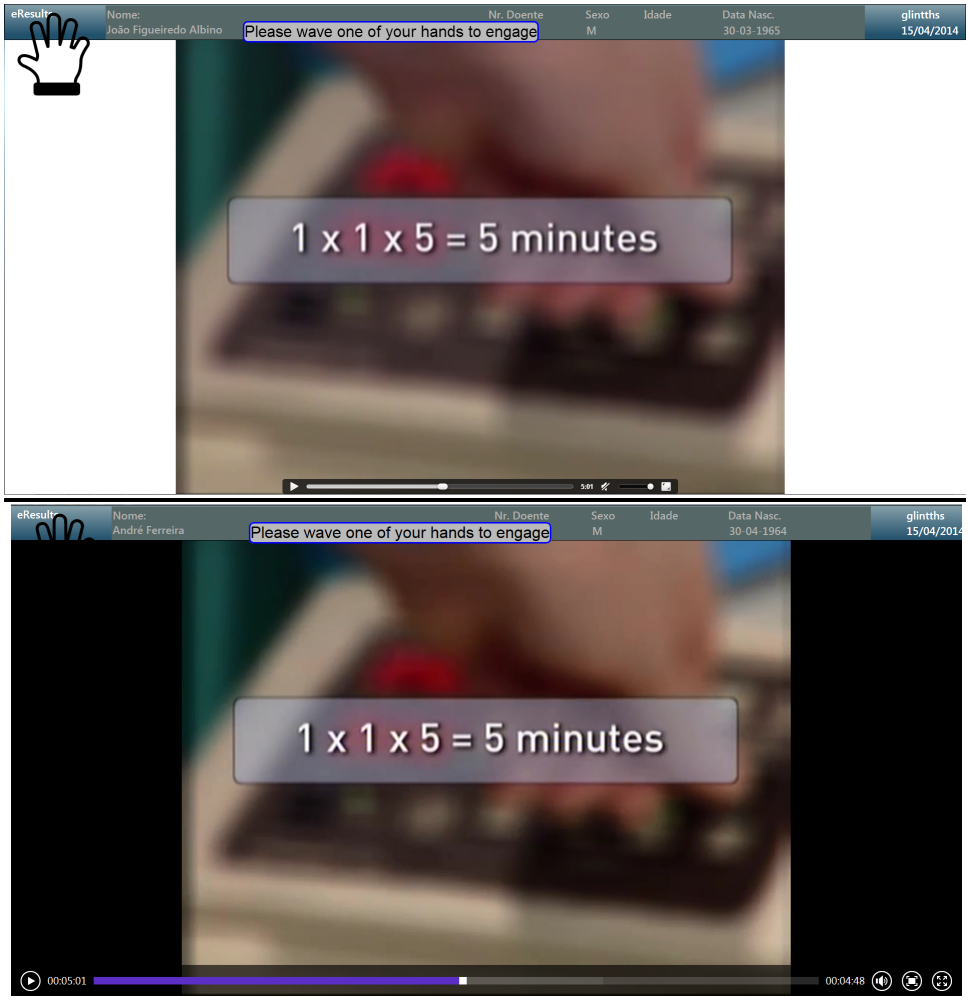

Figure 4.14: Comparison between the display of a video element in Google Chrome (top) and Internet Explorer (bottom).

user who had a false positive grip during document manipulation, making him feel as though this component could be more intuitive. In addition, although the users were reluctant to extend their arms in order to press a button, they later admitted that it was efficient despite initially seeming counter-intuitive.

Table 4.5 shows the means of the results obtained in group B, which had the same global average time as group A, despite having higher average times for image and document manipula-

Component	Time taken (minutes)	Error count	Intuitiveness rating	Tiredness rating
Complete system	3:15	9.4	4.2	4.6
Engagement	0:07	1	4.4	4.4
Button pressing	N/A	0.4	4.4	4.8
Navigation	0:12	0	5	5
Document manipulation	0:24	0.2	4.8	5
Image manipulation	0:19	0	5	5
Video manipulation	1:34	7.8	2.6	4.2

Table 4.4: Average results of the usability tests for the group A.

tions, along with a higher error count. It should be noted that although the navigation and button pressing show a higher error rate than the ones found group A, navigation was generally faster and, since the errors were false negatives, they were not detrimental to the user experience. The button pressing errors were also false negatives and the users in group B found this component to be more intuitive and less tiring than those in group A. Nevertheless, users in group B found their experience with images and documents worse than those of group A and shied away from the use of arm extension for zoom, specially in documents, which have more lag on the zoom operation.

In conclusion, users felt confident and comfortable using the gripped hand as a clutch for all actions and, for image and document manipulation, preferred using manipulative gestures common to the ones they found in other NUIs. Therefore, for optimal user experience, developers should use the default modules for image and document manipulation, where the only drawback will be the requirement of two free hands for manipulation, and the swipe option for navigation, which will not only save space in the lateral menu, but also perform faster. In addition to these results, the result of the individual sessions can be found in the appendix C.

Component	Time taken (minutes)	Error count	Intuitiveness rating	Tiredness rating
Complete system	3:15	12.2	3.8	5
Engagement	0:05	0.8	4.8	4.8
Button pressing	N/A	1.4	4.8	5
Navigation	0:07	0.6	4.6	5
Document manipulation	0:37	1	4	5
Image manipulation	0:30	0.2	3.4	4.4
Video manipulation	1:27	8.2	2.8	4.2

Table 4.5: Average results of the usability tests for the group B.

Chapter 5

Conclusions and future work

This thesis had the goals of solving the problem of computer interaction in the operating room with no-touch interfaces, defining a set of usability rules for these interfaces and implement a web framework to easily adapt existing interfaces to no-touch. By employing manipulative gestures, users managed to complete multiple tasks involving all the relevant use cases in eResults application, plus video manipulation, with an average time smaller than three and a half minutes and all components except video manipulation had at least one solution where, on average, a novice user would only commit one mistake and give the application a score of at least 4.5 in a five-point Linkert scale regarding its intuitiveness and lack of effort required.

Additionally, a framework was implemented making the implementation of no-touch interaction in a web application as simple as marking the components as interactive and calling an initialization function, which allows the developers to create no-touch interfaces without having to create the interaction from scratch.

Nevertheless, this work could see extension and improvements: the video functionality is far from optimal, with users having large problems while attempting to seek within the video, extra functionality such as text input would allow for a better adaptation of eResults and more precise technology such as the upcoming Kinect v2 would reduce the error rate in the application. The following sections suggests how these improvements could affect the implemented work and how to implement them.

5.1 Improvement of the video functionality

The usability results reveal that there is clearly room for further improvement in the video manipulation component and that the main reason for this was the user feedback that it provided. The first step to improve upon this module would be to implement a custom progress bar, universal to every browser and larger than the one found in Google Chrome, in order to make it easier to

observe at larger distances and this progress bar should have a large mark signalling the current time within the video, that the user could drag within the bar to seek within the video.

Afterwards, the reason for the delay in the video and progress bar updates should be studied, as they may be caused due to browser issues, problems caused by the video size or format, which may be fixable by forcing a video format or implementing video buffering, or due to implementation issues.

5.2 Additional functionality

Despite the positive results, the gesture vocabulary largely hinges on the use of the hand grip as an interaction clutch. This limits the expandability of the implemented functionalities: for example, it would be complicated to add a selection function for copy and pasting in images or documents. This can be improved upon by having more precise technology, capable of distinguishing between more hand states than an open or closed hand, or by adding a gesture to switch between operational mode sets, where each mode allows the user to perform different funcions.

Additionally, textual input would also be a valuable addition to no-touch interfaces and, in particular, to a more complete adaptation of eResults. Gestural text input is not recommend because not only is it tiresome, but also because fitting all the letters on the screen with pressable dimensions is quite challenging. Nevertheless, there are some measures that could mitigate these problems:

- An overlay with a circular menu, similar to the one suggested by Chattopadhyay and Bolchini [CB14], where a first-level menu consists of buttons that expand into a second level-menu when the user hovers over them. Figure 5.1 demonstrates a quick mockup of this concept, which would allow the user to use one hand to hover over an inner circle to expand a group of letters, while using the other hand to press the letters;
- Text prediction, similar to the one employed by touchscreen keyboard applications such as SwiftKey could make text input less tiresome, by saving the user button presses. These keyboards attempt to predict the user input, by using a dictionary, and allow them to automatically complete the word and could work well by using the valid possible inputs as the dictionary used for prediction;
- If the keyboard is displayed in a QWERTY layout, an input system method the one found in SwiftKey Flow [Swi12] would also make this system less tiring. With this method, the user is able to press the first letter, maintain the press and move the hand to the other letters in the word, releasing the press at the last letter, and the system will predict the word based on the movements, therefore reducing the necessary time to input a word and making this input less tiresome.

These hypotheses were studied as possibilities for text input but were discarded due to lack of time to properly implement them, as they require advanced work in both interface design and implementation, and in artificial intelligence.

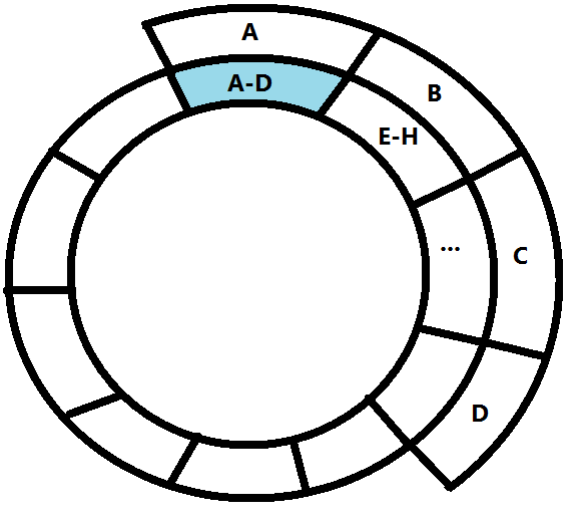

Figure 5.1: Mockup of a radial keyboard

5.3 Improvements brought by the Kinect for Windows v2

The Kinect for Windows v2, which will be publicly available on July of 2014, will allow for expansion on the work done for this thesis. This device will feature a color camera capable of recording at a resolution of 1920x1080 pixels and a depth camera capable of recording at 512x424 pixels [Ash14], which are both higher resolutions than the 640x480 color camera and 320x240 depth camera used for this work. This results in a more fluid and accurate tracking and the capability of distinguishing between more fine grained states for the hands: the beta SDK already features a lasso state [Ash13], corresponding to a gripped hand with the thumb and index fingers extended, which already allows for an expansion in gesture vocabularies, and it is reasonable to expect that more hand states and perhaps even finger tracking will be added to the SDK. Additionally, the SDK has been tweaked and the hand state is no longer tracked in the interaction stream, but rather, it is bound to the skeletal stream [Ren14], which means that grip and grip release events will no longer stop being tracked when the hand leaves the physical interaction zone, which allows developers to use these events in a wider array of situations.

Finally, it should be noted that an adaptation of the native application developed in this thesis should be sufficient to allow to use the developed framework with higher precision and, since the SDK is an evolution of the already existing one, which uses the same concepts of streams and physical interaction zone, this adaptation should be trivial. However, to enlarge the implemented gesture vocabulary with new hand states, the framework will require more extensive changes, by adding new types of messages to the API, creating the appropriate handlers and linking them to the `KinectConnection` module.

References

[Ash13] James Ashley. Kinect for Windows v2 first look. Available at http://www.imaginativeuniversal.com/blog/post/2013/11/26/Kinect-for-Windows-v2-First-Look.aspx, last accessed on June 22, 2014, 2013.

[Ash14] James Ashley. Quick reference: Kinect 1 vs Kinect 2. Available at http://www.imaginativeuniversal.com/blog/post/2014/03/05/Quick-Reference-Kinect-1-vs-Kinect-2.aspx, last accessed on June 22, 2014, 2014.

[Bau11] Nicole Baut. Surgeons use XBox to keep hands sterile before surgery. Available at http://www.thestar.com/life/health_wellness/2011/03/24/surgeons_use_xbox_to_keep_hands_sterile_before_surgery.html, last accessed on January 27, 2014, March 2011.

[Ber13] Jennifer Romano Bergstrom. Moderating usability tests. Available at http://www.usability.gov/get-involved/blog/2013/04/moderating-usability-tests.html, last accessed on February 6, 2014, April 2013.

[Car10] Mary-Jane Caroll. Text legibility and readability of large format signs in building and sites. *Design Resources*, 2010.

[Cat12] David Catuhe. Kinect toolbox. Available at http://kinecttoolbox.codeplex.com/, last accessed on February 6, 2014, July 2012.

[CB14] Debaleena Chattopadhyay and Davide Bolchini. Touchless circular menus: Toward an intuitive UI for touchless interactions with large displays. *AVI 2014 International Working Conference on Advanced Visual Interfaces*, pages 33–40, May 2014.

[Che14] Nancy Chen. The sensor is always on. Available at https://www.leapmotion.com/blog/the-sensor-is-always-on/, last accessed on June 22, 2014, 2014.

[Fai12] Harry Fairhead. OpenNI 2.0 - another way to use the Kinect. Available at http://www.i-programmer.info/news/194-kinect/5241-openni-20-another-way-to-use-kinect.html, last accessed on June 4, 2014, 2012.

[GFG+04] C. Graetzel, T. Fong, S. Grange, , and C. Baur. A non-contact mouse for surgeon-computer interaction. *Technology and HealthCare 12*, pages 245–257, 2004.

[Goo09] Google. Google docs - viewer. Available at https://docs.google.com/viewer, last accessed on June 4, 2014, 2009.

REFERENCES

[Gor12] Michael Gorman. Leap motion gesture control technology hands-on. Available at http://www.engadget.com/2012/05/25/leap-motion-gesture-control-technology-hands-on/, last accessed on February 4, 2014, May 2012.

[GPC11] Luigi Gallo, Alessio Pierluigi Placitelli, and Mario Ciampi. Controller-free exploration of medical image data: experiencing the Kinect. *24th International Symposium on Computer-Based Medical Systems (CBMS)*, pages 1–6, 2011.

[GPP12] Luigi Gallo, Alessio Pierluigi Placitelli, and Giuseppe De Pietro. A kinect nui for 3d medical visualization. *Demonstration competition at ICPR 2012*, pages 1–2, 2012.

[HKR+10] Harshith, Karthik.R.Shastry, Manoj Ravindran, M.V.V.N.S Srikanth, Naveen, and Lakshmikhanth. Survey on various gesture recognition techniques for interfacing machines based on ambient intelligence. *International Journal of Computer Science & Engineering Survey (IJCSES) Vol.1, No.2*, pages 31–42, November 2010.

[Inc07] Adobe Systems Incorporated. Javascript for Acrobat API reference, 2007.

[JOS+11] R. Johnson, K. O'Hara, A. Sellen, C. Cousins, and A. Criminisi. Exploring the potential for touchless interaction in image-guided interventional radiology. *Proceedings of the 2011 annual conference on Human factors in computing systems*, pages 3323–3332, 2011.

[Juh13] Bethany Jean Juhnke. *Evaluating the Microsoft Kinect compared to the mouse as an effective interaction device for medical imaging manipulations*. PhD thesis, Iowa State University, 2013.

[JW13] Mithun George Jacob and Juan Pablo Wachs. Context-based hand gesture recognition for the operating room. *Pattern Recognition Letters*, pages 196–203, June 2013.

[JWP12] Mithun George Jacob, Juan Pablo Wachs, and Rebecca A Packer. Hand-gesture-based sterile interface for the operating room using contextual cues for the navigation of radiological images. *J Am Med Inform Assoc*, pages 183–186, July 2012.

[Kal12a] Pantelis Kalogiros. KinectJS. Available at http://kinect.childnodes.com/, last accessed on June 4, 2014, 2012.

[Kal12b] Pantelis Kalogiros. KinectJS documentation. Available at http://kinect.childnodes.com/docs/, last accessed on June 4, 2014, 2012.

[Ks05] Maria Karam and M.C. schraefel. A taxonomy of gestures in human computer interaction. *ACM Transactions on Computer-Human Interactions*, pages 1–45, 2005.

[Lea13a] LeapMotion. Menu design guidelines. Available at https://developer.leapmotion.com/documentation/cpp/practices/Leap_Menu_Design_Guidelines.html, last accessed on February 11, 2014, 2013.

[Lea13b] LeapMotion. User experience guidelines. Available at https://developer.leapmotion.com/documentation/cpp/practices/Leap_UX_Guidelines.html, last accessed on February 11, 2014, 2013.

REFERENCES

[Lin12] Brad Linder. List of Windows 8 touch-based gestures. Available at http://liliputing.com/2012/03/list-of-windows-8-touch-based-gestures.html, last accessed on June 4, 2014, 2012.

[LM13] Inc Leap Motion. Get started. Available at https://developer.leapmotion.com/documentation, last accessed on January 25, 2014, 2013.

[Mic12a] Microsoft. Kinect for Windows sensor components and specifications. Available at http://msdn.microsoft.com/en-us/library/jj131033.aspx, last acessed on February 4, 2014, 2012.

[Mic12b] Microsoft. Kinect sensor. Available at http://msdn.microsoft.com/en-us/library/hh438998.aspx, last acessed on February 5, 2014, 2012.

[Mic13a] Microsoft. 1.8 SDK and developer toolkit known issues. Available at http://msdn.microsoft.com/en-us/library/dn435682.aspx, last accessed on June 22, 2014, 2013.

[Mic13b] Microsoft. Kinect for Windows | Human Interface Guidelines v1.8, 2013.

[Mic13c] Microsoft. Kinect for windows dev center. Available at http://www.microsoft.com/en-us/kinectforwindowsdev/Downloads.aspx, last acessed on February 4, 2014, 2013.

[Mic13d] Microsoft. Kinect interaction concepts. Available at http://msdn.microsoft.com/en-us/library/dn188673.aspx, last accessed on June 4, 2014, 2013.

[Mic13e] Microsoft. Web applications. Available at http://msdn.microsoft.com/en-us/library/dn435664.aspx, last accessed on June 4, 2014, 2013.

[Moz14] Mozilla. Pdf.js. Available at https://github.com/mozilla/pdf.js/, last accessed on June 4, 2014, 2014.

[NSMG03] Michael Nielsen, Moritz Störring, Thomas B. Moeslund, and Erik Granum. A procedure for developing intuitive and ergonomic gesture interfaces for HCI. *Gesture-Based Communication in Human-Computer Interaction*, pages 409–420, 2003.

[Ope12a] OpenKinect. OpenKinect. Available at http://openkinect.org/wiki/Main_Page, last acessed on February 4, 2014, 2012.

[Ope12b] OpenNI. OpenNI. Available at http://www.openni.org/, last acessed on February 4, 2014, 2012.

[Pte13] Vangos Pterneas. Kinect and HTML5 using WebSockets and canvas. Available at http://www.codeproject.com/Articles/309306/Kinect-and-HTML-using-WebSockets-and-Canvas, last accessed on June 4, 2014, 2013.

[Pte14] Vangos Pterneas. Implementing Kinect gestures. Available at http://www.codeproject.com/Articles/716741/Implementing-Kinect-gestures, last accessed on June 13, 2014, 2014.

REFERENCES

[RB12] Aaron Rothberg and Janet Bailey. Manipulating medical images: A hands-off approach. *Southwest Decision Sciences Institute 43rd Annual Meeting*, pages 1–2, 2012.

[Ren14] David Renton. Kinect v2 the next generation of motion sensing. Presented at Codebits VII, available online at https://videos.sapo.pt/BBDUqcZh4NRMsI82G8dc, last accessed on July 16, 2014, 2014.

[SN12] Nathan Shedroff and Christopher Noessel. *Interaction Design Lessons from Science Fiction*. Rosenfeld, first edition, 2012.

[Ste11] Steve. Kinect SDK Dynamic Time Warping (DTW) gesture recognition. Available at http://kinectdtw.codeplex.com/, last accessed on February 6, 2014, July 2011.

[Str14] Structure. OpenNI 2 downloads and documentation. Available at http://structure.io/openni, last accessed on June 4, 2014, 2014.

[SWE08] Helman I Stern, Juan P. Wachs, , and Yael Edan. Gesture-based communication in human-computer interaction. *2008 IEEE International Conference on Semantic Computing*, pages 1–8, 2008.

[Swi12] SwiftKey. Introducing SwiftKey flow. Available at http://swiftkey.com/en/blog/introducing-swiftkey-flow/, last accessed on June 22, 2014, 2012.

[Vel13] András Velvárt. Kinect interactions with(out) WPF – part iii: Demystifying the interaction stream. Available at http://dotneteers.net/blogs/vbandi/archive/2013/05/03/kinect-interactions-with-wpf-part-iii-demystifying-the-interaction-stream.aspx, last accessed on June 17, 2014, 2013.

[WSE+07] Juan Wachs, Helman Stern, Yael Edan, Michael Gillam, Craig Feied, Mark Smith, and Jon Handler. Gestix: A doctor-computer sterile gesture interface for dynamic environments. *Soft Computing in Industrial Applications Advances in Soft Computing Volume 39*, pages 30–39, 2007.

[WW11] Daniel Wigdor and Dennix Wixon. *Brave NUI World*. Morgan Kaufmann, first edition, 2011.

Appendix A

Native application class diagram

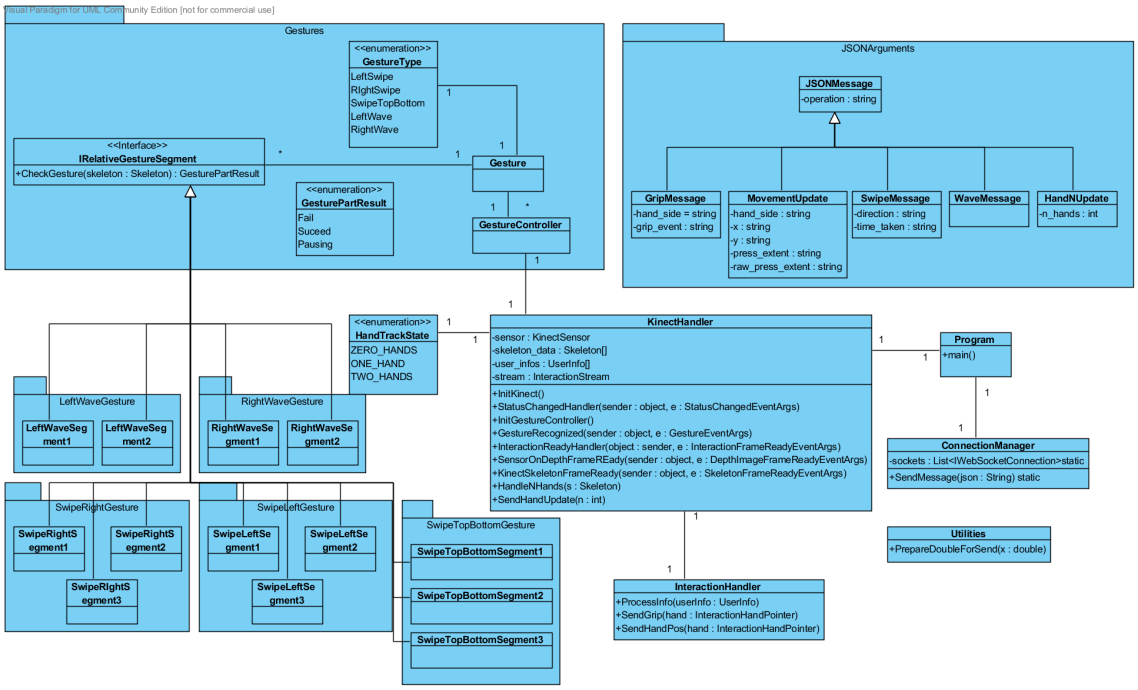

Figure A.1: Class diagram of the native application that processed the Kinect data.

Native application class diagram

Appendix B

Two handed image interaction pseudocode

$distance \leftarrow$ GETHANDDISTANCE
if $(leftHandGripped \ \& \ rightHandGripped)$ **then**
    $translation \leftarrow false$
    $currentSlope \leftarrow \frac{Y_{left} - Y_{right}}{X_{left} - X_{right}}$
    $angle \leftarrow \arctan(\frac{currentSlope - originalSlope}{1 + currentSlope * originalSlope})$
    if $(angle < 0 \ \& \ previousAngle > 1)$ **then**
        $angle \leftarrow \pi - \text{ABS}(angle)$
    else if $(angle > 0 \ \& \ previousAngle < -1)$ **then**
        $angle \leftarrow angle - \pi$
    end if
    $rotationAngle \leftarrow rotationAngle + (angle - previousAngle)$

    $LVDifference \leftarrow leftY - previousLeftY$
    $RVDifference \leftarrow rightY - previousRightY$
    $LHDifference \leftarrow leftX - previousLeftX$
    $RHDifference \leftarrow rightX - previousRightX$
    if $(\text{SIGNAL}(LVDifference) = \text{SIGNAL}(RVDifference) \ \&$
$\text{SIGNAL}(LHDifference) = \text{SIGNAL}(RHDifference))$ **then**
        $translation \leftarrow true$
        $translationY \leftarrow translationY + \frac{(leftVerticalDifference + rightVerticalDifference)}{2}$
        $translationX \leftarrow translationX + \frac{(leftHorizontalDifference + rightHorizontalDifference)}{2}$
    end if
    if $(!translation)$ **then**
        $distanceDifference \leftarrow distance - previousDistance$
        $scale \leftarrow scale + distanceDifference/1000$

```
        if (Scale < 0) then
            scale ← 1/ABS(scale)
        end if
    end if
    TRANSFORMATIONUPDATE(rotation, scale, translationY, translationX)
    previousAngle ← angle
end if
previousDistance ← distance
```

Appendix C

Individual usability test results

C.1 Individual group A results

Component	Time taken (minutes)	Error count	Intuitiveness rating	Tiredness rating
Complete system	3:10	11	4	4
Engagement	0:04	0	5	5
Button pressing	N/A	0	4	4
Navigation	0:10	0	5	5
Document manipulation	0:20	0	5	5
Image manipulation	0:15	0	5	5
Video manipulation	1:47	11	2	3

Component	Time taken (minutes)	Error count	Intuitiveness rating	Tiredness rating
Complete system	4:10	15	4	5
Engagement	0:10	1	4	4
Button pressing	N/A	1	4	5
Navigation	0:13	0	5	5
Document manipulation	0:19	0	5	5
Image manipulation	0:22	0	5	5
Video manipulation	2:21	13	2	4

Component	Time taken (minutes)	Error count	Intuitiveness rating	Tiredness rating
Complete system	2:52	5	5	5
Engagement	0:12	3	4	4
Button pressing	N/A	0	5	5
Navigation	0:10	0	5	5
Document manipulation	0:29	0	5	5
Image manipulation	0:15	0	5	5
Video manipulation	0:57	2	3	5

Component	Time taken (minutes)	Error count	Intuitiveness rating	Tiredness rating
Complete system	2:56	8	4	4
Engagement	0:04	0	5	5
Button pressing	N/A	1	4	5
Navigation	0:11	0	5	5
Document manipulation	0:25	1	4	5
Image manipulation	0:19	0	5	5
Video manipulation	1:18	6	3	4

Component	Time taken (minutes)	Error count	Intuitiveness rating	Tiredness rating
Complete system	3:07	8	4	5
Engagement	0:07	1	4	4
Button pressing	N/A	0	4	5
Navigation	0:18	0	5	5
Document manipulation	0:27	0	5	5
Image manipulation	0:26	0	0	5
Video manipulation	1:30	7	3	5

C.2 Individual group B results

Component	Time taken (minutes)	Error count	Intuitiveness rating	Tiredness rating
Complete system	3:06	8	4	5
Engagement	0:04	0	5	5
Button pressing	N/A	0	5	5
Navigation	0:07	0	4	5
Document manipulation	0:44	2	3	5
Image manipulation	0:40	1	3	4
Video manipulation	1:02	5	3	5

Component	Time taken (minutes)	Error count	Intuitiveness rating	Tiredness rating
Complete system	3:07	14	4	5
Engagement	0:09	2	4	4
Button pressing	N/A	2	4	5
Navigation	0:07	1	5	5
Document manipulation	0:26	1	4	5
Image manipulation	0:33	0	3	5
Video manipulation	1:18	8	3	5

Component	Time taken (minutes)	Error count	Intuitiveness rating	Tiredness rating
Complete system	3:56	22	3	5
Engagement	0:04	0	5	5
Button pressing	N/A	3	4	5
Navigation	0:06	0	4	5
Document manipulation	0:32	0	5	5
Image manipulation	0:24	0	4	4
Video manipulation	2:23	19	1	3

Component	Time taken (minutes)	Error count	Intuitiveness rating	Tiredness rating
Complete system	3:04	9	4	5
Engagement	0:03	0	5	5
Button pressing	N/A	2	5	5
Navigation	0:08	1	5	5
Document manipulation	0:42	1	4	5
Image manipulation	0:29	0	4	5
Video manipulation	1:13	5	4	4

Component	Time taken (minutes)	Error count	Intuitiveness rating	Tiredness rating
Complete system	3:02	8	4	5
Engagement	0:10	2	5	5
Button pressing	N/A	0	5	5
Navigation	0:07	1	5	5
Document manipulation	0:42	1	4	5
Image manipulation	0:22	0	3	4
Video manipulation	1:21	4	3	4