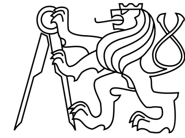
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF BACHELOR'S THESIS

Title:	SAGElab - gesture driven control using Kinect 2.0
Student:	Tomáš Kasalický
Supervisor:	Ing. Ji í Chludil
Study Programme:	Informatics
Study Branch:	Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2016/17

Instructions

The goal of the project is to design and implement a contactless gesture driven control system based on the Kinect technology for SAGElab (the wall of displays with resolution 9600 x 4320px each). The system will allow basic tasks such as slide presentation control, window resizing, positioning, etc.

1. Conduct a research of existing Kinect-based contactless control solutions and describe their functionality.
2. Analyze systems of gesture driven control designed for various purposes and describe them.
3. Design a new or improve an existing user-friendly solution for SAGElab control.
4. Implement an application prototype for gesture driven control for the display wall in the SAGElab.
5. Perform appropriate tests on the resulting solution.

References

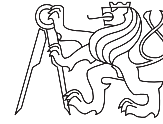
Will be provided by the supervisor.

L.S.

Ing. Michal Valenta, Ph.D. Head of Department	prof. Ing. Pavel Tvrdík, CSc. Dean

Prague February 6, 2016

Czech Technical University in Prague
Faculty of Information Technology
Department of Software Engineering



Bachelor's thesis

SAGElab - gesture driven control using Kinect 2.0

Tomáš Kasalický

Supervisor: Ing. Jiří Chludil

15th May 2016

Acknowledgements

I would like to thank my thesis advisor, Ing. Jiří Chludil, for his valuable advice and help he provided me with. I would also like to thank the administrators of SAGElab, who were very helpful when I needed any feedback on my ideas. Lastly I would like to thank my family for their support during my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 15th May 2016 . . . .

Czech Technical University in Prague
Faculty of Information Technology
© 2016 Tomáš Kasalický. All rights reserved.
This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Kasalický, Tomáš. *SAGElab - gesture driven control using Kinect 2.0*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

Cílem této bakalářské práce bylo prostudovat existující technologie využívající hloubkové senzory a detekci gest a aplikovat získané vědomosti při návrhu řešení ovládání gesty pro vizualizační prostředí SAGE2 za pomocí zařízení Microsoft Kinect 2.0. Návrh řešení bral v úvahu reálné nasazení v SAGElab, síťové a multimediální laboratoři na Fakultě informačních technologií Českého vysokého učení technického v Praze. V práci byl kladen důraz na vytvoření přirozeného uživatelského rozhraní za použití gest rukou. Na základě návrhu byl vytvořen prototyp aplikace, který byl prověřen testy použitelnosti pro získání zpětné vazby z reálného použití. Implementované návrhy obdrželi pozitivní hodnocení ze strany uživatelů a ukázaly velký potenciál této technologie. Výsledek této práve je velmi komplexní prototyp aplikace bezdotykového ovládání SAGE2 a stěny displejů v SAGElab, clusteru 20 displejů o celkovém rozlišení 9600 x 4320 px. Výsledky experimentu také poskytují hodnotnou zpětnou vazbu pro mou budoucí práci v oblasti počítačového vidění a interakce člověka s počítačem a také pro práci všech výzkumníků a studentů kteří se zajímají o tuto oblast.

Klíčová slova Interakce člověka s počítačem, přirozené uživatelské rozhraní, ovládání gesty, sledování pohybu, počítačové vidění, hloubkový senzor, bezdotykové ovládání, vizualizace ve vysokém rozlišení, hloubkový snímek.

Abstract

The goal of this bachelor's thesis was to research current technologies which use depth sensors and gesture detection, and to collect interesting ideas and apply them to design a gesture control solution for SAGE2 with the use of Microsoft Kinect 2.0. The design was taking into consideration a practical usage of SAGE2 in SAGElab - networking and multimedia laboratory at administrated by CESNET and FIT and FEL at Czech Technical University in Prague. The focus while designing the solution was to create a natural user interface with the use of hand gesture detection. To verify the usability of the suggested solution, I created an application prototype, implemented according to the before mentioned design, and performed usability testing on the prototype to receive feedback from practical usage. The ideas implemented in this solution received a positive feedback from the users and overly performed well. The main outcome of the testing was, that this technology has a lot of potential and even the prototype application showed insteresting results. The result of this thesis, is a very complex prototype application, which enables users to control SAGE2 and thus the display wall in SAGElab, a cluster of 20 displays with total resolution of 9600 x 4320 px. The results of the experiments provide a valuable feedback for my future work in the area of computer vision and human-computer interaction and for future work of other researchers and students interested in this area.

Keywords Human-computer interaction, natural user interface, gesture control, motion detection, computer vision, depth sensor, contactless control, high resolution visualization, depth image.

Contents

Introduction **1**

1 Existing solutions **3**
1.1 Previous SAGElab Solution[1] 3
1.2 Non-SAGE Solutions 4

2 Analysis **9**
2.1 Requirements 9
2.2 Depth Camera 11
2.3 The Room Setup 13
2.4 Gesture Analysis 13
2.5 SAGE2 Actions 22

3 Design **27**
3.1 Presentation Setup 27
3.2 Interaction Setup 31
3.3 User Calibration 36
3.4 Software Design 36

4 Implementation **41**
4.1 Code Style 41
4.2 Loading Configuration 41
4.3 Initialization & Main Loop 42
4.4 Gesture Detection 42
4.5 SAGE2 Communication 45
4.6 Gesture & Actions Mapping 45
4.7 Graphical User Interface 47

5 Testing **51**
5.1 Test Scenarios 51

5.2 Performance Evaluation 54
5.3 Results 55

6 Future Work **57**
6.1 Gesture Improvements 57
6.2 Additional Functionality 58

Conclusion **61**

Bibliography **63**

A Acronyms **67**

B Contents of enclosed CD **69**

List of Figures

1.1 Mouse button click gestures in Lukáš Sedláček's thesis.[1] 4

2.1 Depth frame from Microsoft Kinect 2.0.[2] 11
2.2 Depth camera measurements.[3] 12
2.3 Presentation setup with camera facing the display wall 14
2.4 Presentation setup with camera on the side of the display wall . . 14
2.5 Pointing to the screen gesture 15
2.6 Grab and drag gesture 17
2.7 Flick gesture 18
2.8 Span gesture 20
2.9 SAGE2 web interface 22

3.1 Presentation setup 28
3.2 Presentation setup: switching applications gesture 30
3.3 Feedback monitor: switching applications gesture 30
3.4 Interaction setup 31
3.5 State diagram: interaction modes 33
3.6 State diagram: interaction mode switching 34
3.7 State diagram: windows interaction 35
3.8 Window resizing gesture 35
3.9 T-pose 37
3.10 Deployment diagram 38

4.1 Point gesture to cursor position converstion 46
4.2 Web based graphical user interface 49

5.1 Initial setup for testing 52
5.2 Initial setup for presentation mode testing 54

List of Tables

2.1 Kinect devices comparison[4] 12
2.2 Gesture accuracy testing 21

5.1 Gesture rating and error counts 55

Introduction

In today's technologically advanced world, we interact with technology and computers every day, and for many users the interaction with devices still is a challenge which prevents them from enjoying their experience with any gadgets. Very frequently, we use new technology to help us improve our life and to make our work easier. If we really want our equipment to make our life easier, it needs to be easy to manipulate, without a need for extensive learning and getting used to unintuitive user interfaces. Unfortunately, for many years, the technology was presented as if it was made only for people who are willing to spend all their effort and time researching the technology. Because of that, for many years, software was not designed well for a regular user. Today, the situation is different. The technology, in a form of smartphones and other smart devices, has become a part of most of the people's lifes which caused the developpers to focus on the regular user much more than ever before.

There still are technologies, wich are not commonly used, and need to be improved in terms of their usability. One such technology is the high resolution display wall in SAGElab[5], a multimedia laboratory at Faculty of Information Technology at Czech Technical University in Prague[6]. The display wall is a cluster of 20 displays with total resolution of 9600 x 4320 px. The entire screen measures 5.58 meters in diagonal[5]. Big screen like this requires a very specific control interface. The device runs on the SAGE[7] environment (SAGE2[8] respectively), which is a software developed specifically for data visualization on large, high resolution screens. The main features of the system are visualization of various applications in separate windows, each application can show data visualizations, slide presentation, high resolution pictures, shared desktop and many others. To some extent the SAGE environment resembles a very light operational system. The software allows the user to control the environment from his or her personal computer using mouse and keyboard. This is not always the most convenient way to do it. The big screen is very frequently used for team work, or presentation in front of a crowd of people, and this requires the user to interact with the screen directly. Even

a solo work with high resolution data or multiple windows often requires the user to be able to interact with the visualization environment without having to use a regular computer in-between.

The topic of this thesis is to solve the problem mentioned in the previous paragraph. My main task is to design a usable solution for controlling the display wall using gestures with a help of a depth camera, more specifically, Microsoft Kinect 2.0[9]. Depth camera is a device capable of capturing the depth image of a scene, which is picture which contains an information about the distances of each of the points in the picture. It can be used to recognize objects in the scene more accurately than in a regular rgb image. This allows us to use the depth camera to track the user and to recognize the user's motion and gestures.

The main goal of this work is to come up with a usable gesture control software for the SAGE2 environment. SAGE2 is the new version of the SAGE environment, that is being developed by the Electronic Visualization Laboratory[10] at University of Chicago[11]. The outcoming software will be available to anyone using SAGE2 and the main requirement on the result is to be intuitive and easy to use.

Chapter 1

Existing solutions

Purpose of this section is to analyze gesture driven solutions used for various purposes. This analysis will help me get an overall idea of the techniques used in gesture driven controls. I will also collect some of the gestures used in these systems and used them when coming up with my own system for SAGE. Since my solution has to be intuitive I plan on reusing common gestures used in various systems, which will help the user feel familiar with the interface and it will make the system feel more intuitive.

1.1 Previous SAGElab Solution[1]

First solution I need to address is a solution of my previous colleague - Lukáš Sedláček, who worked on a contactless control for sage in SAGElab before me. He created a solution protype as a part of his bachelor's thesis. This solution maps motion of a hand to cursor movement and three different gestures, which employ both user's hands represent the left, right, and middle mouse button click. The intention of the solution was to cover any possible action by mapping the mouse behaviour to hand gestures.

The implementation itself works correctly, but the software is not used by anyone in the SAGElab. The reason is, that the solution is very difficult to learn and even when learned, it makes any task very difficult to complete. The gesture for the cursor movement is simple, it tracks the hand and maps the hand's movement to a cursor movement. In this system, hands are labeled as a mouse hand, the first one detected, and the gesture hand, the second detected. The left mouse button click is represented by moving the gesture hand in front of the user. The right hand click is accomplished by moving the mouse hand in front of the user and the middle button click is represented by moving the gesture hand forward with a small offset to the side. Scrolling the mouse wheel is done by moving both hands forward and moving them away from, respectively towards each other. These gestures are visualized in figure 1.1.

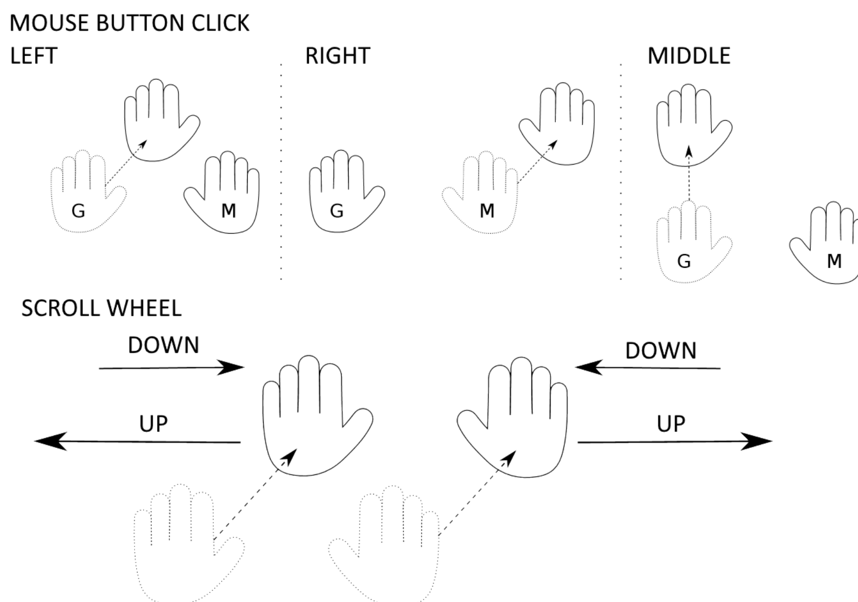


Figure 1.1: Mouse button click gestures in Lukáš Sedláček's thesis.[1]

1.2 Non-SAGE Solutions

Since SAGE and SAGE2 are not very commonly used systems, there are no other Kinect based solutions for these systems. Therefore I need to analyze solutions created for other purposes as well.

1.2.1 Official Kinect 2.0 Gestures for Xbox One[12]

This is the solution created by Microsoft for the game console Xbox One. Microsoft is the manufacturer of Kinect 2.0, so their solution should be a valuable source of inspiration.

This gesture driven control mostly maps hand gestures to cursor actions. These are the gestures detected by this software:

Cursor movement The position of the cursor is mapped to the position of the user's hand. This is a straight forward solution and it feels very natural to the user.

Cursor click Moving the hand forward and then back. This gesture is simulating the feeling of pressing a button. The disadvantage of the gesture is, that when pushing the hand forward, the hand can be moved to the side and move the cursor away from the button the user intends to click.

Zoom For zooming, the user clenches his fist, as he was grabbing the screen, and by moving the hand back or forward zooms the screen in or out. This gesture feels very natural, since it really feels like the user was grabbing the screen and moving with it.

Scroll Very similar gesture to the zoom gesture. By moving the hand to the sides or up and down, the content on the screen moves in the desired direction. This also feels very intuitive, since simluates the feeling of grabbing the screen and moving with it.

Opening context menu Moving the hand forward and staying in that position for a while, about two seconds, opens the context menu.

1.2.2 Kinect for Windows[13]

This is another solution created by Microsoft, designed for use with personal computers. This solution also maps hand gestures to control the cursor.

Cursor movement Same as in previous solution, the position of the cursor is mapped to the user's hand position.

Left mouse button click Gesture for this is performed by moving the hand quickly forward and back.

Left mouse button double click This gesture is almost the same as the previous one, only the hand is held in the forward position for about two seconds.

Right mouse button click This is simulated by the opposite of the left mouse button click gesture, by moving the hand back and forward again.

Drag and drop To grab an item and drag it across the screen, the user has to move the hand back, this will grab the item, move the item across the screen and move the hand back forward, to release the item.

1.2.3 Win&I[14]

This is a small solution developed to be used on personal computers. This application also maps the gestures to control the mouse cursor.

Cursor movement The cursor position is mapped to the hand position. There also is an additional gesture, when user performs a flick, the mouse cursor moves towards the edge of the screen in the direction of the flick.

Left mouse button click This is simulated by holding the cursor above a clickable element for a certain amount of time. There is a timeout running on the screen to have a feedback on whether the user is holding

the cursor above the element correctly. This system might get very inconvenient, when the number of clickable elements on the screen gets higher.

1.2.4 Gestigon[15]

This solution seemed pretty advanced, but it is designed mostly for a regular sized personal computer screen.

Cursor movement The position of the mouse cursor is determiend by the position of user's hand and eyes. The application calculates a line from the user's eyes, through the finger tip towards the screen, and the point where this line intersects the screen, is where the cursor is moved to. So to the user, it feels like the cursor is always where the user points to using his or her finger. When the cursor reaches an edge of the screen, the screen scrolls towards this direction.

Left mouse button click Mouse button clicked is performend by finger tap in the air.

Navigation Quick wave in some direction simulates navigation forward or back(right or left), which can be used in slideshow presentation, or for example when browsing the web, to return to the previous page.

Context menu Holding an open hand in one position and then rotating the hand quickly to vertical position opens the context menu.

Zoom Moving the hand away from the screen performs a zoom out.

1.2.5 TV Nova - Shaman[16]

Shaman is a software developed by the czech commercial tv station Nova. It utilizes the kinect depth sensor and it is used for the weather forecast program. The newscasters use various gestures to switch between the individual sections of the forecast. Unlike most common weather forecast programs, the visualizations are shown in front of the newscasters so they do not have to interrupt the eye contact with the viewers.

A lot fo the gestures take into account where the user currently is in the final picture. The weather casters can see the image along with the graphics that are added into it, so they have feedback where in the imagine they currently are.

Switching graphics To get the next weather forecast graphics, the user moves his or her hand outside of the image, and simulates a movement of pulling the new graphics from outside of the picture. To move the graphics away from the screen, the user simulates a movement of pushing the graphics out off the screen.

Show additional information To show an additional information, such as temperature, the user waves their hand steadily from the top of the screen down.

Highlighting an element To highlight an element on the screen, the users simply point at the desired element.

Chapter **2**

Analysis

This chapter consists of several steps. First, is the analysis of the requirements on the resulting solution. The following section analyzes the depth camera - the device I will be using in this solution. After the hardware analysis, I will analyze the possible setup of the sensor within the SAGElab, or any other area where my resulting solution could be used. The Kinect placement will be based on the general idea of what tasks we need to use the device for. Afterwards, I will analyze the possible gestures I am capable of detecting, this analysis will help me choose and combine the best performing and most intuitive gestures. The last part of my analysis will look into the various tasks allowed by SAGE2 to the user. After the last part, I will have all the parts I need to map the gestures to the tasks to design the complete solution.

2.1 Requirements

This section of the analysis describes the requirements on the gesture control system. This is a set of functionalities and characteristics, the resulting solution needs to have. These requirements were collected while consulting with my thesis advisor and with the students activelly involved in SAGElab.

Software requirements are usually split into two main groups, nonfunctional requirements and functional requirements.

The collected requirements will be taken into account while designing the solution to ensure, the outcoming application will meet the needs of the users.

2.1.1 Nonfunctional Requirements

These are the required criteria, which do not include some specific behavior but they restrict the system and define some certain quialities the solution needs to have. [17]

This is the list of the nonfunctional requirements:

N1 Kinect 2.0 compatibility The application needs to utilize Microsoft Kinect 2.0, the most recent version of the Kinect depth sensor.

N2 Compatible with SAGE2 The solution needs to be compatible with SAGE2 environment, which is used for the display wall in the SAGElab.

N3 Placeable in SAGElab The design of the setup of the room has to allow this system to be used within SAGElab. It also needs to take in account a regular usage of the lab, so the placement of the sensor and other hardware needs to allow a regular traffic in the room.

N4 Configurable The solution needs to be configurable from a configuration file. This file needs to contain mainly a configuration of the SAGE2 server hostname and port.

N5 Natural user interface The solution needs to be a natural user interface. The solution needs to implement the hand gesture in such a way, they will feel natural to the users. This will be measured by testing the usability of the outcoming solution.

2.1.2 Functional Requirements

This section describes the specific required behavior of the application. The functional requirements generally described the functionalities the users need from the application and the behavior of these functionalities.[18]

F1 Hand gesture detection The application needs to base its functionality on hand gesture detection.

F2 Window positioning and resizing The application needs to allow the user to move and resize the windows within SAGE2 using the hand gestures.

F3 Controlling SAGE2 applications The solution needs to allow the user to control the main functionalities of the SAGE2 applications. These are the applications and the required functionalities that needs to be controlable using hand gestures:

- Google maps, Deep viewer, Sketchfab browser - These applications need to allow the user to move the content such as maps, a picture or a 3D model inside the window, and zoom the content in and out.
- PDF viewer - This application needs to allow the user to switch pages of the PDF document.
- Video player - This application needs to allow the user to play and stop the video.

F4 Slideshow presentation The application needs to allow the user to perform a slideshow presentation with a usage of purely hand gestures to control it. The gestures need to allow the user to flip through PDF slides and play videos.

2.2 Depth Camera

This section describes the device I will be using, Kinect 2.0 and the technology of depth cameras in general.

Figure 2.1: Depth frame from Microsoft Kinect 2.0.[2]

Depth camera is a device, which as a regular camera captures pictures of the scene, but the pictures coming out of the depth camera do not contain the color information, but they contain an information of the depth in the scene. Each pixel of the picture contains an information about how far (deep) the point in the picture is from the camera. The picture can be visualized as a gray scale picture, where the depths in the picture are transformed into shades of gray. An example of a depth frame is in figure 2.1.

Most of the common depth cameras use the *Time of Flight* technology.[19] This technology commonly uses the technique of shooting infrared light rays into each point of the scene, and calculates the distance of the particular point from the time the light ray took to return back to the sensor, using the information that the infrared light ray travels at speed of light.

In figure 2.2, you can see what the depth measurements in the depth frame represents. The light rays measure the distance from the sensor, but since the sensor is aware of the angle the ray was shot in, it can calculate the distance from the plane of the camera and not from the camera itself. So if the camera is parallel with a screen, the distance returned by the sensor is always the distance from the plane of the screen.

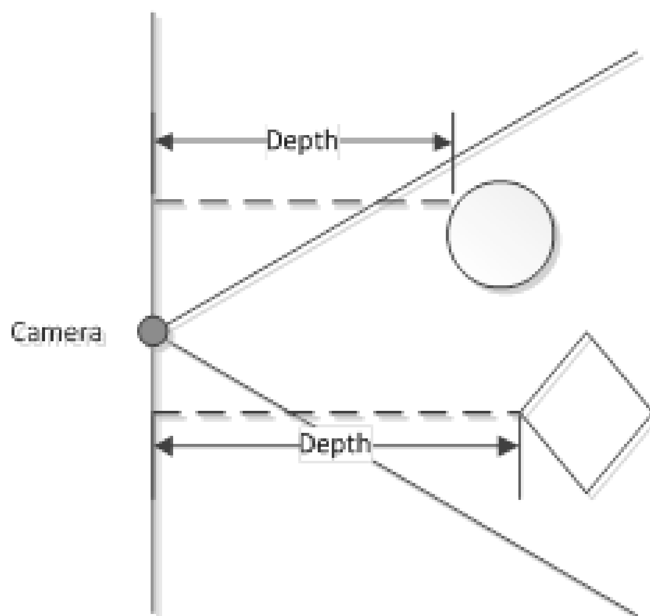


Figure 2.2: Depth camera measurements.[3]

2.2.1 Microsoft Kinect 2.0

This is a combined device developed by Microsoft, and it includes a depth sensor and an rgb camera. This hardware was originally designed to be delivered along with the XBox gaming console, but it is also sold separately and it is possible to be connected to a personal computer.[9]

In table 2.1, you can see comparison of parameter of Kinect 2.0 and the previous version of Kinect.

Table 2.1: Kinect devices comparison[4]

Device	Depth camera resolution	RGB camera resolution	Horizontal field of view	Vertical field of view
Kinect 2.0	512 x 424	1920 x 1080	70°	60°
Kinect	320 x 240	640 x 480	57°	43°

2.2.2 Skeleton Tracking

The depth frame is just a raw data, which is not a very valuable information. So to retrieve some more useful information from the depth sensor, I will need to use libraries for user skeleton tracking. This is the standard procedure when developing gesture detection systems. The libraries possible to be used with Kinect 2.0 are either a Microsoft Kinect 2.0 SDK[20] or a NiTE[21]

library. The output from processing the depth frames using these libraries is a skeleton, which is a data structure that contains positions of the main joints in the user's body.

I can use this data to detect various gestures by tracking the movement and position of each joint, mostly by tracking the joints of the user's hands.

2.3 The Room Setup

The room setup and the position of the sensor is very important when designing the solution. The display wall, which I am developing this solution for, is used in two main scenarios, slideshow presentation and interaction with SAGE2 applications.

2.3.1 Interaction Setup

This is the setup where the user wants to interact directly with the applications and their content. This setup needs to allow the user to manage the windows in SAGE2 and to manipulate the applications.

When the user wants to interact with the applications, he or she is most likely to stand facing the display wall, because the user wants to look directly at the managed content. For this scenario the best place for the depth camera is right bellow the display wall facing the user.

2.3.2 Presentation Setup

This setup is more complicated for gesture detection. The user is most likely to stand facing the audience, so the user will stand in front of the display wall or right next to it. For this purpose, a camera placed bellow the display wall will not be very efficient. The options are to put the camera facing the presenting person or have the camera to the side of the user.

The first option, when the camera is facing the user and the display wall, would require the camera to be placed bellow the ceiling. This option is visualized in figure 2.3. This setup also contains a feedback monitor visible to the user, so the user does not need to look at the display wall to see what actions he is performing.

The second option is to place the Kinect sensor to the side of the user. In this setup, the user would be facing the Kinect sensor, standing on the side of the display wall. This setup can also contain a feedback monitor on the side of the display to be visible to the user.

2.4 Gesture Analysis

In this section I will analyze the individual gestures, I am capable of detecting using the depth camera. These gestures will be inspired by the solutions

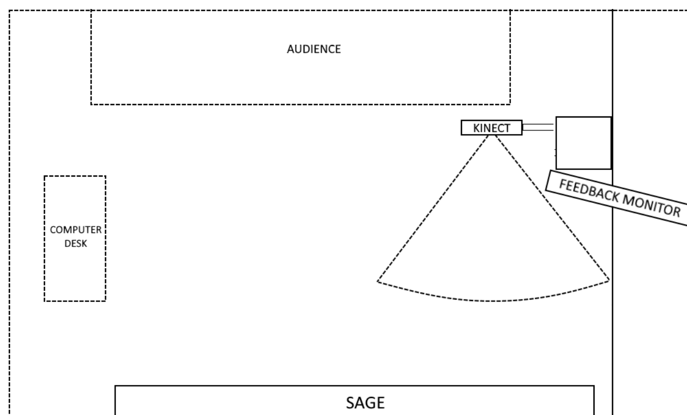


Figure 2.3: Presentation setup with camera facing the display wall

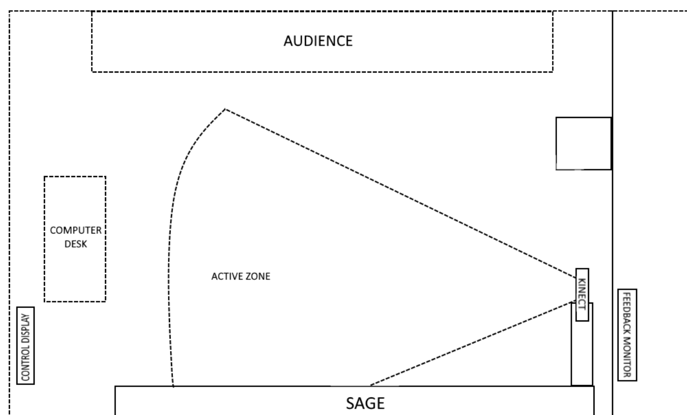


Figure 2.4: Presentation setup with camera on the side of the display wall

analyzed in chapter 1 and other control systems, such as touch control. Each gesture will have description with an included illustration and a mention of where the gesture is most commonly used. The description will be followed by analysis of the limitations we have to be aware of when using the gesture, such as the possible angle of detection and the accuracy of detection.

The accuracy of the gesture's detection will be tested experimentally on prototype algorithms. These prototype algorithms will perform a very simplified gesture detection. Since the final detection algorithms can be significantly different from the prototypes, this testing will only be an estimate of the gesture detection accuracy. Some of these algorithms have parameters such as time, distance or speed. These parameteres will be chosen by trial and error and will have very little statistical foundation. This should be sufficient for the estimation purpose.

2.4.1 Pointing to the Screen

2.4.1.1 Description

This is an easy gesture, where the user simply points to the screen and the spot he is pointing to can be used as a position for the cursor or position for element selection. Similar gesture was used in the bachelor's thesis of Lukáš Sedláček, where the cursor movement was mapped on the movement of users hand. The gesture can be improved by taking into account the position of user's eyes. If we connect the position of user's eyes, position of his or her hand we define a line. The intersection of this line with the display wall gives us an exact point on the screen where the user is pointing to. This solves the issue of the previous bachelor's thesis, where there was no correlation between where the user was pointing to and position of the cursor.

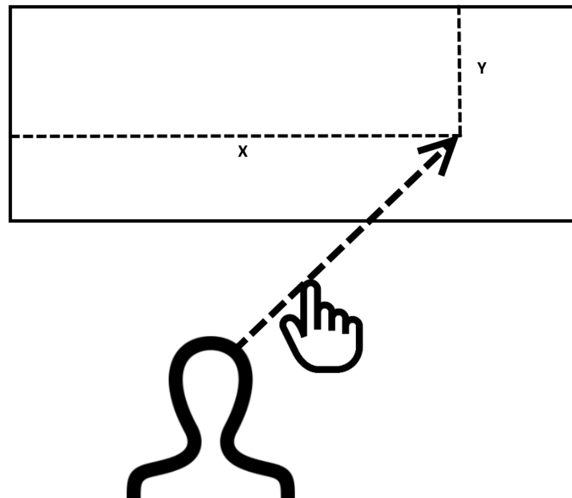


Figure 2.5: Pointing to the screen gesture

2.4.1.2 Limitations

This gesture is best to be used when standing opposite the display wall, with the depth sensor placed bellow the screen. This way the user is pointing directly against the depth camera which will make the calculation more accurate (Interaction setup). The gesture can also be used when the sensor is facing the display and the user is pointing to the display from a side (Presentation setup), in this setup we need to guess the exact window, if there is more than one window intersecting with the point gesture. In the presentation mode, this gesture might be easier to detect if we put the depth sensor to the side from the display and the user, this way the camera will give us more accurate depth measurements.

2.4.1.3 Testing Algorithm

The testing algorithm will be tracking the position of the head of the user and the position of the right hand. From these two positions we calculate a vector, where the user points, the vector will be calculated as follows:

$$pointingVector = handPosition - headPosition$$

Where $pointingVector$, $handPosition$, and $headPosition$ are 3D vectors. If the $pointingVector$'s length exceeds 50 cm, and the $handPosition$ is at least 20 cm closer to the sensor than the $headPosition$, the algorithm detects the point gesture. These conditions simulate the detection of the arm being fully stretched and pointing anywhere in front of the user. The point where the user is pointing will be calculated as the intersection of the display plane and a line which has a direction of $pointingVector$ and goes through the $headPosiion$ point.

2.4.2 Pan

2.4.2.1 Description

This gesture is performed by dragging an elemnt on screen and moving, dragging, it across the screen by the hand movement. To get the accurate position of the object being dragged, we can use the technique from previous the point gesture, which uses combination of hand position and the position of user's eyes. Other than moving an element on the screen, this gesture can also be used to flip pages of a document or switch slides in a slideshow presentation and anything similar to these actions.

To recognize when a grab is intended, we can detect a fist clench, which will represent the grab. Another option is to detect when the user's arm is entirely straighten, which will mean the user is reaching for the element on the screen.

2.4.2.2 Limitations

The main limitation of this gesture is the grab detection. Detecting a clenched fist is only implemented in the official Microsoft Kinect SDK[20] and is not very reliable, since when standing opposite the depth sensor, clenched fist and open hand appear on the depth image very similar. Also when using other libraries than the official Kinect SDK, I will have to implement the fist detection myself, since there is no open source implementation available.

The angle limitations are very similar to the point gesture, the most suitable mode is the interaction setup. The presentation setup will make it very difficult to use this gesture since we cannot detect the point the user is pointing to accurately.

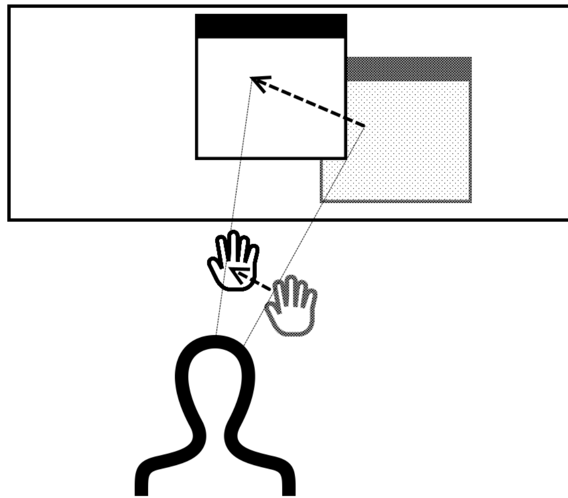


Figure 2.6: Grab and drag gesture

2.4.2.3 Testing Algorithm

There is no trivial algorithm to detect a clenched fist, so this gesture will not be tested. The main part of this gesture is the same as the point gesture, so to have an estimate of the possible detection accuracy of this gesture, I will use the results of the point gesture testing.

2.4.3 Flick / Wave

2.4.3.1 Description

The flick gesture is partially taken from a regular touch control interface. It is usually used to flip between pages or slides. The gesture is perform simply by waving hand in one direction, but to detect when the gesture is intended correctly, some required speed of the movement must be reached. We can divide this into 4 separate gestures by the direction of the movement. This gesture can easily be detected as long as the user is facing the depth camera.

I can also take inspiration from the TV Nova's gesture system, Shaman[16]. They use a steady wave movement from top to bottom to show next set of information. This can be used in the presentation setup, since it is very similar to the weather forecast in TV Nova's Shaman. The user can stand next to the display wall and by performing this movement, next slide can appear. To go back one slide, the same movement in reverse can be used.

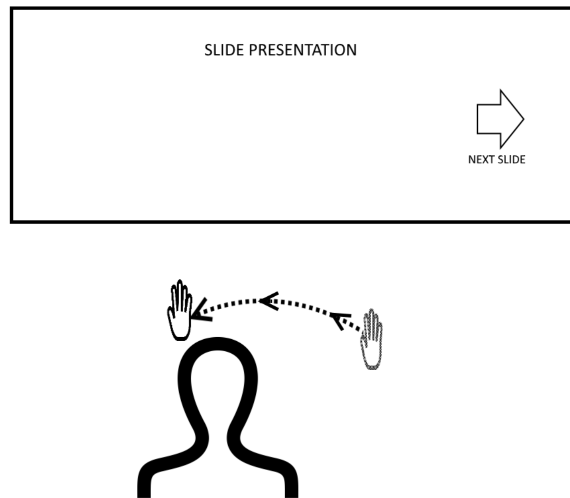


Figure 2.7: Flick gesture

2.4.3.2 Limitations

The biggest issue of this gesture is to define the required speed for it to be detected as flick. If we choose speed too slow, it will be detected when performing other gestures, such as the grab and drag. But we need to choose speed, that will the user be physically capable of reaching.

2.4.3.3 Testing Algorithm

The testing algorithm for this gesture will be tracking a movement of the hand and whenever the hand travels at least 40 cm in the time bellowe 0.8 s, the wave gesture will be detected. As the direction of the gesture, the average movement direction will be calculated and then the algorithm will check, to which direction out of left, right, up, and down is this direction the closest.

2.4.4 Raise Hand

2.4.4.1 Description

Raising users hand can be user as a notification gesture. It can be used for the user to let the software know about some activity the user just wants to start. It is performed simply by lifting user's hand above his or her head. This gesture is used by the Xbox gaming console.

2.4.4.2 Limitations

The main limitation of this gesture will be its possible accidental detection when performing other gestures. Whenever user performs any gesture with

hands above head, it can unintentionally trigger this gesture. To avoid this problem, when combining the gestures together, I have to set up the parameters of this gesture experimentally, to make sure this gesture is not detected too often.

2.4.4.3 Testing Algorithm

Testing the raise hand gesture will be very simple. Whenever the user puts his or her hand at least 10 cm above the head, and leaves it at this position for at least 0.6 s, the raise hand gesture will be detected.

2.4.5 Push / Move Hand Forward and Back

2.4.5.1 Description

This gesture has two different versions, one is moving hand in direction against the depth sensor and the second one is to move the hand to the side in presentation mode, which can create effect of pushing some element off the screen. The gesture can have some variations. The user can just simply move his hand forward or backward to signal some event, to make the gesture more obvious for the detection we can add a movement back to original position. So for instance the user can move hand forward and back to signal some event, such as a mouse click. A variation of this gesture was also used in Lukáš Sedláček's thesis, where a combination of both hands was used, to make the gesture more variable.

2.4.5.2 Limitations

The main problem of this gesture is to set the parameters correctly. The required distance of the movement must be enough for it not to be detected unintentionally, but not to be too uncomfortable for the user. Also we have to expect the fact that user will always create some unintended movement to the sides while performing the movement forward.

2.4.5.3 Testing Algorithm

Detection of this gesture will start by detecting a moment, when the user holds a hand in one position for at least 0.5 s. To eliminate small position changes, there will be a tolerance of 10 cm. After holding the hand in one position for at least 0.5 s and then moving the hand forward (respectively backwards) at least 20 cm and then moving the hand back to the original position, the gesture will be detected.

2.4.6 Span

2.4.6.1 Description

This gesture is inspired by common touch control. On a mobile phone or any other touch controlled device, finger span is often used to zoom in and zoom out, or to scale an element. This gesture is essentially the same as the touch gesture, but instead of using fingers, the user will perform this gesture with the help of whole arms.

When moving the hands away from each other, zoom in or scale up action will be triggered and the opposite when closing the hands towards each other.

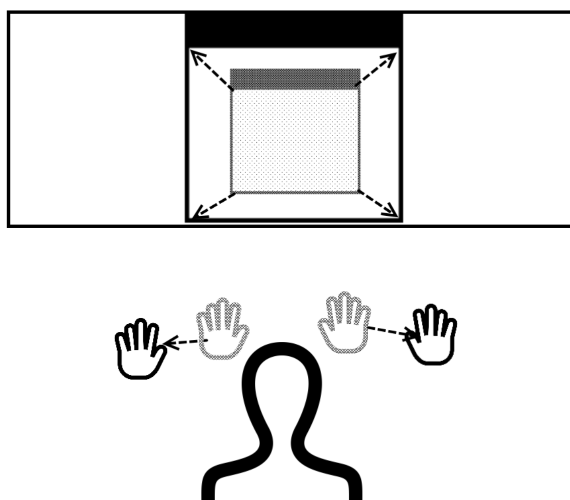


Figure 2.8: Span gesture

2.4.6.2 Limitations

The problem of this gesture will be similar to the grab gesture, because we need to detect correctly, when user wants to trigger this gesture. We can simlarly to the grab gesture calculate, when the arms are straighten. Fully straight arms will trigger the span gesture.

2.4.6.3 Testing Algorithm

This gesture will be detected using the same algorithm as for the point gesture, but using the algorithm for both hands. When a point gesture for both hands is detected at the same time, it is considered to be the span gesture.

2.4.7 Detection Accuracy

To define which gestures are the most reliably detected, I performed detection accuracy testing using the prototype algorithms. I have asked 5 random students, to perform these gestures in front of the kinect sensor. Each test subject was asked to performed each gesture ten times. In table 2.2, we can see the results of the accuracy testing. The first column contains the names of the gestures, second column contains positive detection accuracy in percent. A gesture is positively detected, when it was recognized at the moment the user intended to perform the specific gesture. In the last section of the table, you can see the number of false detection. A false detection is a detection of a gesture while some other gesture was intended to be performed.

Table 2.2: Gesture accuracy testing

	Gesture	Detected	False detections								
			1	2	3	4	5	6	7	8	9
1	Wave up	35	x	7.5	0	0	65	0	7.5	0	28
2	Wave down	30	45	x	0	0	70	0	0	7.5	18
3	Wave left	75	0	0	x	1	85	7.5	0	0	0
4	Wave right	65	0	0	10	x	80	0	0	0	0
5	Point / Pan	90	0	0	0	0	x	15	15	15	0
6	Span	72.5	0	0	0	0	68	x	2.5	2.5	0
7	Push forward	47.5	0	0	0	0	88	5	x	20	0
8	Push back	47.5	0	0	0	0	90	0	40	x	0
9	Raise hand	80	53	50	0	0	75	5	0	0	x

We can see, that a very well performing gesture is the point gesture, but at the same time, this gesture has the most false detections while performing other gestures. So this gesture is not very suitable to be combined with other gestures.

The span gesture was also very accurate. The span gesture is basically the same as the point gesture, performed using both hands at the same time, so it likely for these two gestures to have similar detection accuracy.

The wave gestures performed very inconsistently. The wave left and right had higher detection accuracy than the wave up and down.

The push back and push forward gestures did not perform very well. Also when the push back gesture was recognized, the push forward gesture was falsely detected as well.

The raise hand gesture performed very well and except for the wave up and wave down, it did not trigger many other gestures to be falsely detected. This gesture is very reliable to be used and it can also be efficiently combined with other gestures.

2.5 SAGE2 Actions

This section describes possible actions, which SAGE2 allows. The listed actions were put together with the help of the regular users of SAGE2. I have grouped these tasks into two categories, *windows interaction* and *application interaction*.

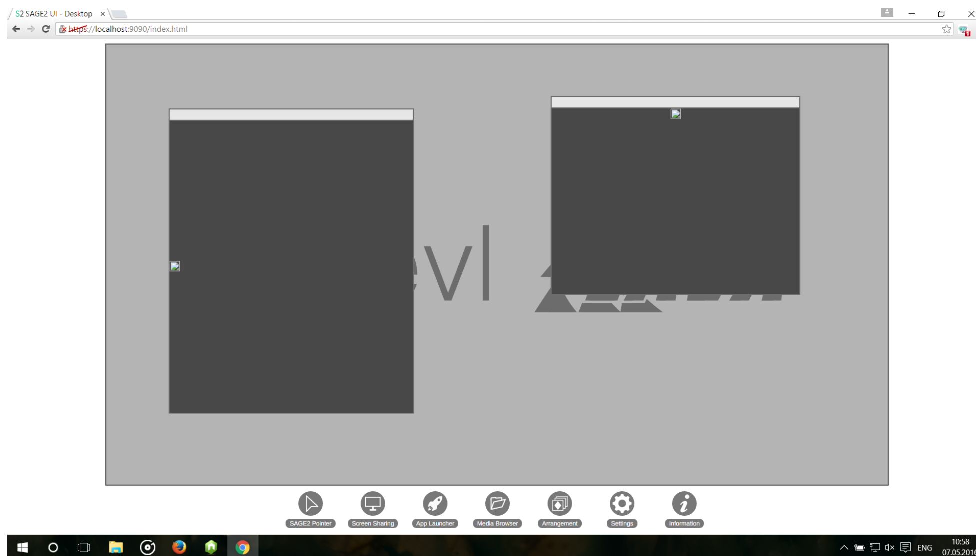


Figure 2.9: SAGE2 web interface

All of the actions are available from the user interface, accessible using a web browser on a computer, this user interface is shown in figure 2.9. The user can control some of the actions from the browser directly, or the user can turn on the SAGE pointer, which creates a cursor in the SAGE2 environment, and this cursor is controlled using the personal computer's mouse.

2.5.1 Windows Interaction

The first group of actions is windows interaction. Using these actions, the user is able to manipulate windows of the applications running on SAGE2.

2.5.1.1 Context Menu

Anywhere on the screen a context menu can be opened using the SAGE pointer and the right mouse button. The context menu opens at the position of the pointer, and it contains several elements circled around the pointer. This context menu can be opened inside the application as well and each application can add custom buttons to it.

2.5.1.2 Starting Applications

The SAGE2 in default supports several applications, and new applications can be developed and installed. The installed applications can be opened from a menu inside the web based user interface. Selected application is opened in a new window on the screen.

2.5.1.3 Closing Applications

The applications opened as described in the previous section must be possible to be closed somehow. This is regularly done by clicking on the top right corner of the application window, or by using a right click and clicking on the close button in the context menu.

2.5.1.4 Positioning

The application windows can be freely moved around the screen. This is performed by a simple mouse drag across the screen. The user clicks on the application window and holds the mouse button down and as he or she moves the pointer across the screen, the window follows the pointer's position.

2.5.1.5 Resizing

The windows can also be resized. The size is changed by scrolling the mouse wheel while the pointer is above the desired window, or by draging the right bottom corner of the window.

2.5.1.6 Switch to Application Interaction

The windows interaction mode can be switched to the applications interaction using the key combination *SHIFT + TAB*.

2.5.2 Applications Interaction

In this mode, the user interacts with the capabilities of the applications directly. The windows can still be manipulated, but if the pointer is inside the inner application area, it manipulates the content of the application.

2.5.2.1 Slideshow Presentation

Slideshow presentation can be shown using the PDF reader inside SAGE2. Possible actions:

- Previous slide - left arrow key.
- Next slide - right arrow key, spacebar.

2.5.2.2 Image Viewer

There are 3 different applications for viewing images in SAGE2. First is a regular image viewer, which allows to show one single picture and does not allow any actions other than positioning the window. Second is a *Photo Slideshow*, which circles through the pictures uploaded to the media folder in SAGE2, and the third one is the *Deep Viewer*, which was developed to show high resolution pictures, these are the actions user is enabled to do in Deep Viewer:

- Zoom in - mouse wheel forward.
- Zoom out - mouse wheel backward.
- Move - mouse pan, arrow keys.

2.5.2.3 Google Maps

This application shows the maps from Google and allows the following operations:

- Zoom in - mouse wheel forward.
- Zoom out - mouse wheel backward.
- Move - mouse pan, arrow keys.

Apart from the above mentioned tasks, the application also offers the following functionalities in the context menu:

- Show traffic
- Switch between regular map and satelite map
- Show weather
- Zoom in / out
- Search for an address - which is typed in using the keyboard

2.5.2.4 Video Player

SAGE2 contains a simple video player, with these actions:

- Play / Stop - spacebar
- Move to certain time - available in context menu using a slider

2.5.2.5 Sketchfab Browser

The SAGElab is often used to visualize 3D models in high resolution. The 3D viewer used in the SAGElab has these possible actions:

- Rotate the model - mouse pan
- Move the model - SHIFT + mouse pan
- Zoom - mouse wheel
- Switch between models - keys 0 - 9

2.5.2.6 Generic Application

There can also be many other applications developed, in most of these application the mouse pan is the main controlling element. There are also application such as *Notepad* and *Sticky Notes*, which enable to user to write down notes using a keyboard, but these are not possible to be controled using gestures.

2.5.2.7 Switch to Windows Interaction

Switching back to windows interaction is performed the same way as switching into the application interaction, using the combination of keys *SHIFT + TAB*.

Chapter 3

Design

This chapter describes the suggested solution and designs all of the individual aspects of the application. The design needs to provide all of the important information I will need to implement this solution. The first part of the chapter describes the functionalities from the user's point of view and it describes how will the outcoming solution behave in specific situations. Second part of this chapter describes the software design, which is a part of the software development process, and it designs how will the solution work from the software developer's propective.

3.1 Presentation Setup

As mentioned in the analysis, one of the scenarios, we need to support, is a slideshow presentation. In this scenario, the user is facing the audience having the display wall behind the presenter's back. In this scenario, the required actions are controlling the slideshow presentation, showing pictures, or playing a video.

3.1.1 The Room Setup

This section will describe the setup of the room, where this control solution will be used. The design is strongly influenced by the dispositions of the SAGElab at Czech Technical University, but possible universal usage will be considered while designing the solution.

The suggested setup is displayed in figure 3.1. The presenter will be standing at the left side of the display wall. The Kinect sensor will be on the opposite side facing the user. So when the user is facing the camera, he is sideways to the audience. When the user wants to control the presentation, he or she needs to stand in the area named *active zone*. This zone will purposely be limited to a certain distance, to avoid accidental detection. In SAGElab, the best distance limit is about 3 meters, the whole display wall is about 4 meters

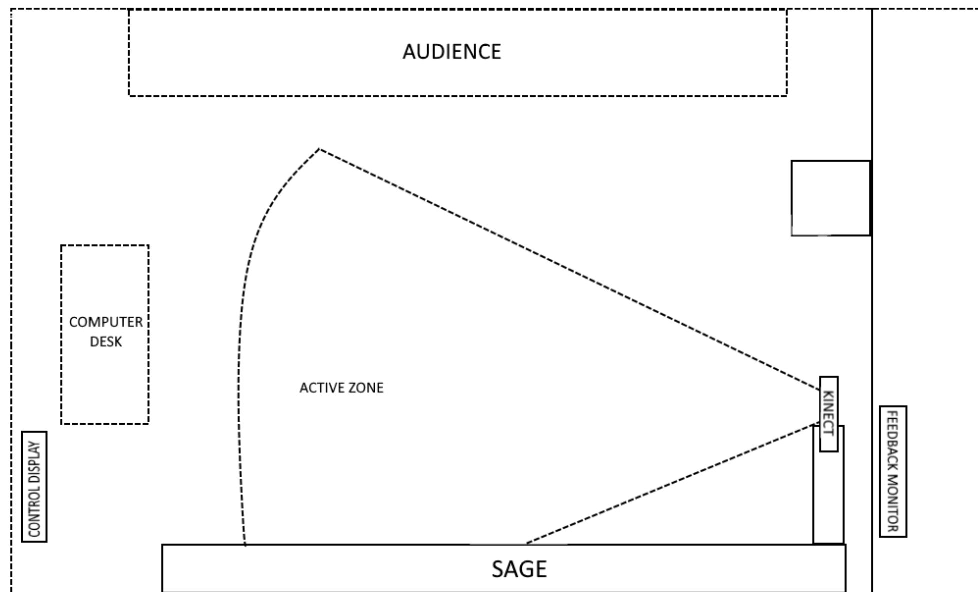


Figure 3.1: Presentation setup

wide. This *active zone* will be indicated by distinctive marks on the floor, these marks will always make it clear whether the user is in the detectable area.

From the gesture testing conducted in my analysis, I have discovered, that what the users were missing the most during the testing was a feedback while performing the gestures. Therefore, I have decided to place a feedback monitor on the side, behind the kinect sensor. In this position, the monitor will be hidden from the audience while being visible to the presenter.

3.1.2 Actions & Gestures

This section will list the actions which will be covered by gestures in the presentation mode and which gesture will be mapped to these actions. As described in the previous section, these gesture will only be detected when the user is in the active zone of the kinect device.

Slideshow Presentation:

- Next slide
- Previous slide

Video Player:

- Maximize

- Play
- Stop

Any application:

- Maximize
- Switch to some other application(or back to slideshow)

Since positioning and resizing of the windows is not necessary during the presentation, all of this has to be prepared before starting the presentation, during the presentation mode, the user will always be in the application interaction mode will the one selected application. There is also no need to show the SAGE2 pointer.

3.1.2.1 Slideshow Presentation

To control the slideshow presentation, I have decided to use the wave left and wave right gestures (illustrated in figure 2.7). Wave left will go to the next slide and wave right will go back to previous slide. I have chosen these two gestures, because they did fairly well during the detection accuracy testing, and because these gestures feel intuitive for flipping through slides or pages.

3.1.2.2 Video Player

To control the video player I will use exactly the same gestures as for the slideshow presentation. I will use the wave left gesture to play the video and wave right to stop the video. Using the same gestures will make the control application easier to learn and also these two gestures are intuitive in the case as well.

3.1.2.3 Maximizing & Switching Applications

The main purpose of this functionality is to allow the presenter to show a video or an image or other figure in the middle of the main slideshow presentation. To make this task simple to perform, I will switch between the applications and maximize the newly selected application during one action. The gesture for this action is visualized in figure 3.2. At the begining of a slideshow presentation, the PDF reader will have the focus. To switch to a different application these steps has to be followed:

1. Raise hand above user's head.

   At this point the space between the top of the hand and the user's hips will be separated into several horizontal fields. There will be as many fields as there is applications running on SAGE2 (without the slideshow presentation application).

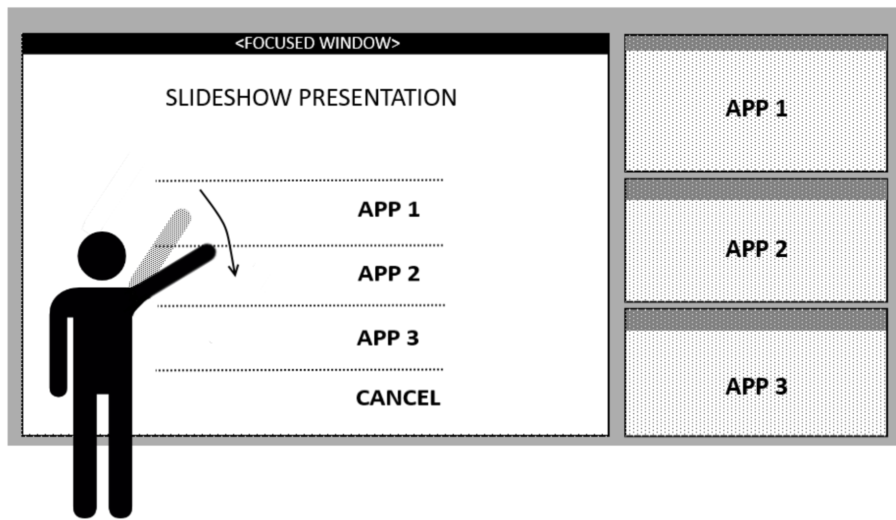


Figure 3.2: Presentation setup: switching applications gesture

2. Position hand in the section of the desired application.

   To provide feedback what application is the user currently selecting, there will be a list of applications shown on the feedback monitor (illustrated in figure 3.3). In the list the currently selected application will be marked with an icon of a hand.

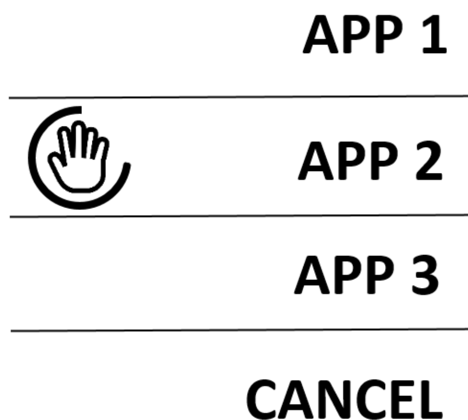


Figure 3.3: Feedback monitor: switching applications gesture

3. Wait for the application to get selected.

   To prevent from accidentally selecting a different application, there will be some minimal time, the user has to hold the hand in the selection position. This time will be visualized on the feedback monitor. This is illustrated in figure 3.3, where the time limit is represented by the circle around the hand icon. This circle will be disappearing in a circular motion and once the whole circle disappeared, the application is selected.

After selecting, the application will switch its position, with the currently focused application. This will represent the maximalization of the application. If initially the slideshow window covers most of the screen, any newly selected application will go into this position. To switch back to the presentation, the same action has to be performed.

The applications in the switch menu will be sorted by their vertical position. The applications can be hidden behind the main window or positioned anywhere on the screen. The layout shown in figure 3.2 is not mandatory.

3.2 Interaction Setup

In this setup, the user interacts directly with the SAGE2 and its applications. In this scenario, the user is standing facing the display wall.

3.2.1 The Room Setup

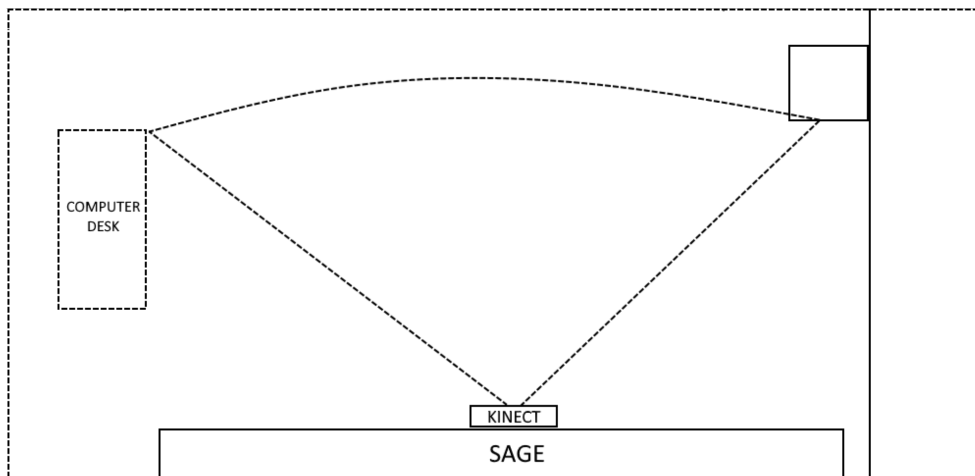


Figure 3.4: Interaction setup

The setup of the room in the interaction mode is shown in figure 3.4. The Kinect sensor is placed in the middle, under the display wall, this way

the kinect is in the best position to detect the user's actions. There is one possible issue in this setup, which is the sensor's angle of view. When the user is on one side of the room, close to the display, the user will be outside of the detectable area. But, I expect the user to stand further away from the screen when trying to interact with it, because this way, the user will have a good overview of the workspace.

3.2.2 Actions & Gestures

3.2.2.1 Mode Switching

I have decided to have three modes in the interaction setup:

Detection Deactivated This mode is the default one, when the user is initially detected. In this mode, the gestures are not detected to prevent from accidentally controlling the SAGE2. Only gestures which are detected are the gestures to switch to the other two modes.

Windows Interaction In this mode, in addition to switch to other modes, the user can change position and size of the windows on the screen and also close the applications.

Application Interaction In this mode, the user can interact with the content of the applications. The interaction will always be restricted to one specific window. This restriction will prevent the user from accidentally interacting with other applications. The actions for the specific applications will be described further in this chapter.

The interaction modes and switching between them are visualized in state diagram in figure 3.5.

There will be two gesture combinations to switch between the modes:

Raise Hand When the user performs the raise hand gesture, if the current mode is *Windows Interaction*, the mode is switched to *Detection Deactivated*. If the current mode is not *Windows Interaction*, this mode is activated.

Raise Hand & Point This gesture combination will always activate the *Application Interaction* mode from any other mode. If the user points at any window on the screen while having one hand raised, the *Application Interaction* for that window is switched on. If the user points somewhere else than on an application window, no action is performed.

The detection of the gesture combinations will have several states, which are visualized in state diagram in figure 3.6.

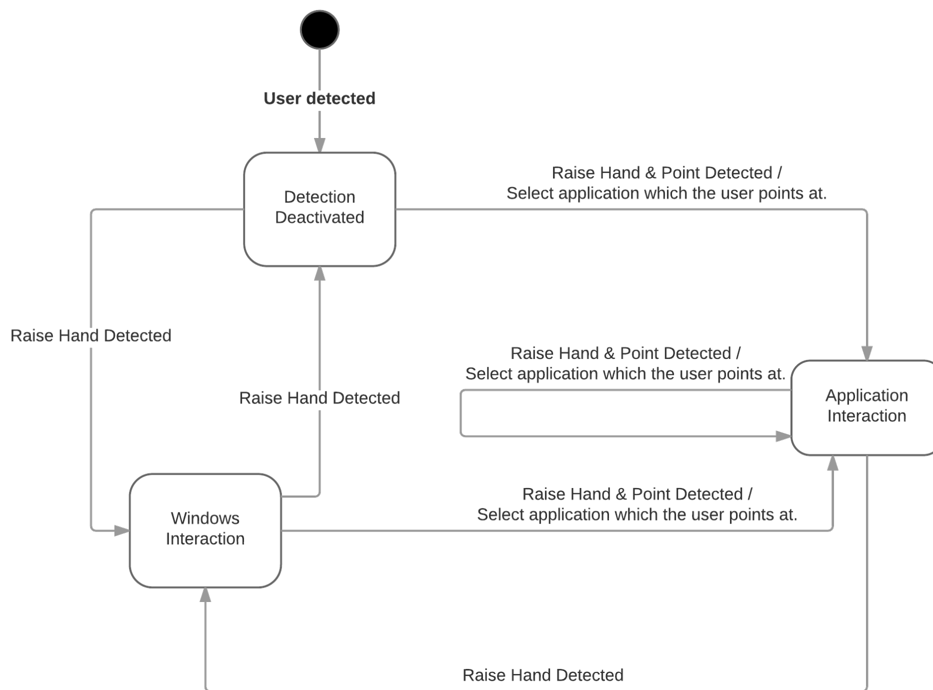


Figure 3.5: State diagram: interaction modes

3.2.2.2 Window Positioning

Changing the position of the windows will be done using the point gesture. This gesture will simulate the movement of grabbing and dragging a window on the screen. The grab of a window will be detected when the arm is fully stretched and pointing to the window. A fully stretched arm will be detected based on the distance of the hand from the shoulder. This distance will be determined by a user calibration described earlier.

To make the positioning more accurate, there will be an invisible grid, the windows will attach to. The size of the grid cells will be defined in a configuration file.

3.2.2.3 Window Resizing

The window resizing will use a combination of point gestures for both hands. Fisrt, the user points at a certain window, this will select this window for resizing, then the user points at screen using his or her second hand, and moving the second hand will resize the window. If the second hand is a right hand, the point on screen where the hand points will be a new position of

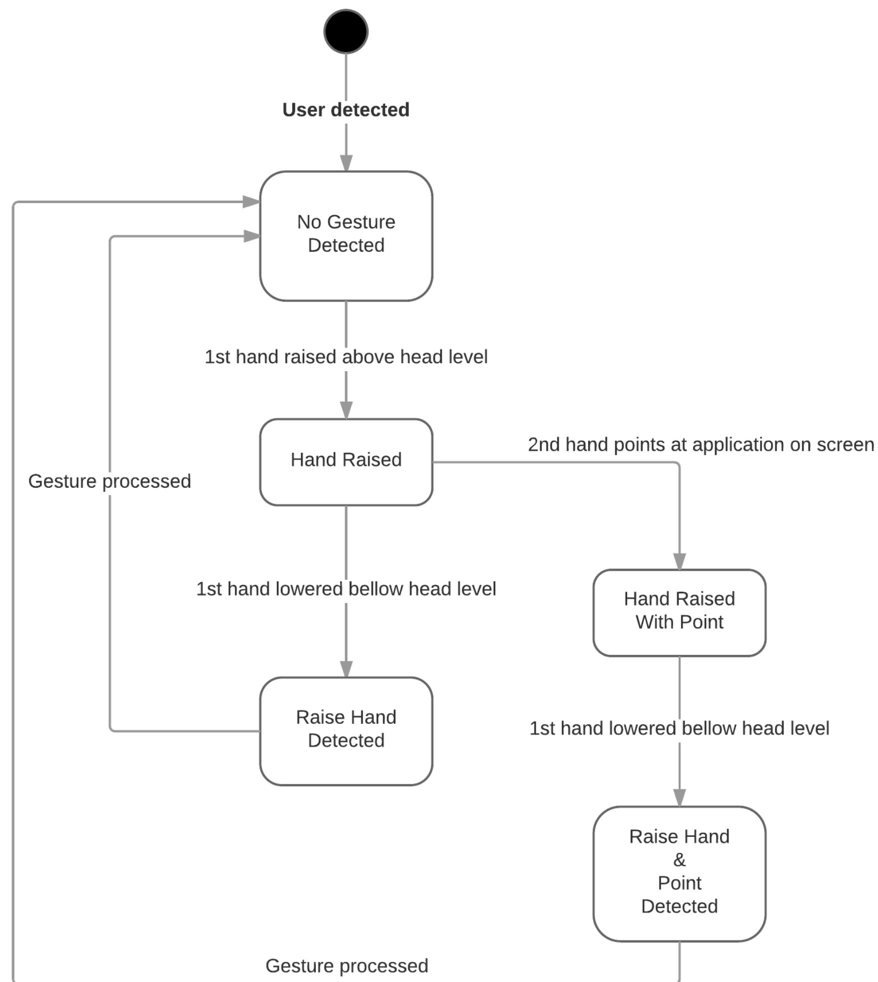


Figure 3.6: State diagram: interaction mode switching

the window's bottom right corner. When the second hand is a left hand, this applies to the bottom left corner. The position of the window stays the same, while the corner of the window is moving with the second hand.

This will allow the user to change the window size arbitrarily. The window size will attach to an invisibile grid the same way as in window positioning. While resizing the first hand can be lowered. The resize gesture will be active until the second hand is not lowered as well. This gesture is visualized in figure 3.8

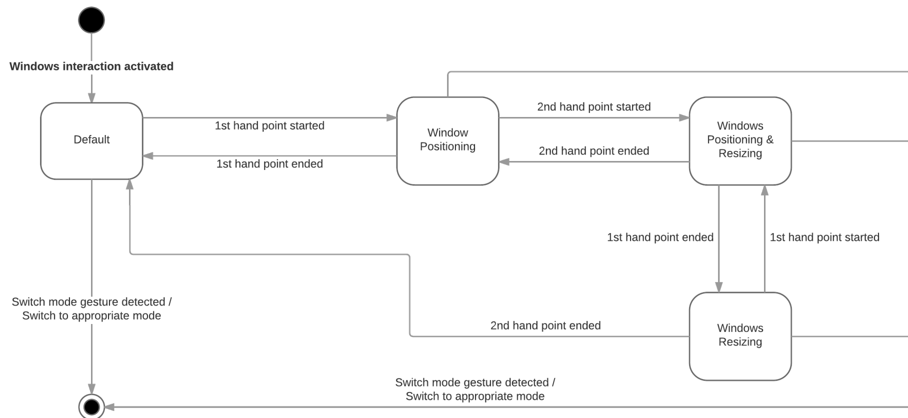


Figure 3.7: State diagram: windows interaction

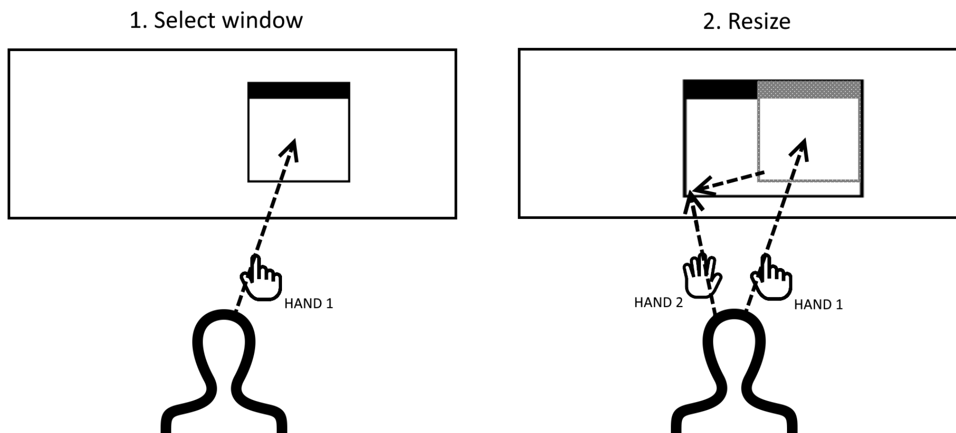


Figure 3.8: Window resizing gesture

3.2.2.4 Generic Application

Most of the applications need two main functionalities, the mouse pan and zoom. For the applications not mentioned bellow in this section, only these two gestures will be recognized. The pan gesture will be implemented using the point gesture. The range of the pan will be limited to the currently activated window. For the zoom, the span gesture will be used.

3.2.2.5 PDF Reader

In interaction setup, the PDF reader will be controlled exactly the same way as a slideshow in the presentation setup. The wave left gesture will flip to the next page and wave right will flip to previous page. The pan and zoom are not necessary in this application.

3.2.2.6 Video Player

The gestures for video player will be the same as in the presentation setup. Wave left will start the video and wave right will stop the video.

3.2.2.7 Sketchfab Browser

Sketchfab browser is an application to view 3D models in SAGE2. In this application, the regular mouse pan is used to rotate the model, while to move the model in the window, SHIFT + mouse pan is used. The rotating and zoom will be mapped to the common gestures - point and span.

3.3 User Calibration

Most of the gesture I use in my solution use some dimensions in their detection algorithms. But these dimensions should differ according to the user. To have all of the dimensions of the user, I will measure the user's arm lengths and overall size using a calibration.

This calibration will not be required. When the user does not perform the calibration, there will be a unit set of dimensions, based on average human size. When the user finds the detection inaccurate, he or she can perform a calibration gesture, to calibrate the system to his or her size.

The calibration will be triggered by the user through an user interface or a command. To calibrate, the user has to stand in a T-pose. This is a pose, where the user stands straight up with arms spread out to the sides. The T-pose is visualized in figure 3.9. The calibration algorithm will measure the distance between each of the joints of the user and save this data. The calibrations will be saved permanently and any saved calibration can be loaded when necessary.

3.4 Software Design

In previous sections, I have designed the individual gestures and behaviour of the application, this section describes design of the application components and suggest solutions which will be used in the implementation.

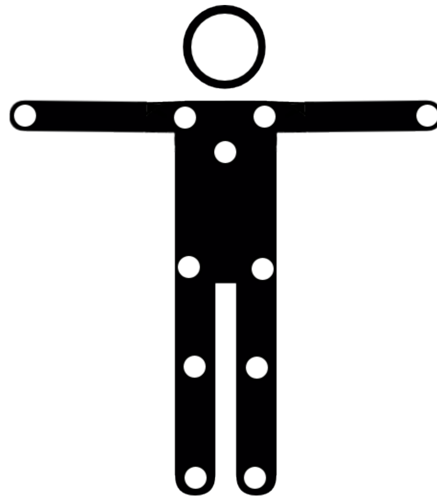


Figure 3.9: T-pose

3.4.1 Components

The application will consist of several components, where each of them has a different purpose. Splitting the software into components makes the link between particular parts of the application weaker and makes it easier to modify the software in the future.

Gesture Detection Application The main component of the application will be the gesture detection application. This component will receive input from the Kinect 2.0 device and process it. The output of this detection will then be mapped to actions which will be sent to the SAGE2 server.

Graphical User Interface Another important component is the graphical user interface. As I found out during the testing in the analysis, a feedback on the user's gestures makes it much easier for the user to learn the to perform the gestures correctly. The feedback will be provided using a graphical user interface. After consulting with the SAGElab administrators, the best solution of the *GUI* should be independent on the main gesture detection component. The gesture detection application will most likely run on a dedicated computer which will not necessarily have a computer screen. Therefore the GUI should be accessible remotely. There are two different GUI versions required.

- SAGE2 Application - Very often, the user will interact with the display wall directly. In this case there will be GUI shown directly in the SAGE2 environment.

- Generic UI - In some cases, for instance in the presentation mode, the GUI on the display wall would be distracting. In these cases, there will be a possibility to show the GUI on a different screen.

The SAGE2 application has to be written in Javascript[22], because the only application programming interface provided by SAGE2 supports Javascript only[23]. To be able to reuse most of the GUI implementation, the generic UI will be also based on Javascript. The javascript code will be included in a simple html page. This has two additional advantages, html and Javascript is platform independent and can be run on any device with a web browser, and also these two technologies were designed to work remotely so the UI can communicate with the main application over network, therefore these components will be fully independent.

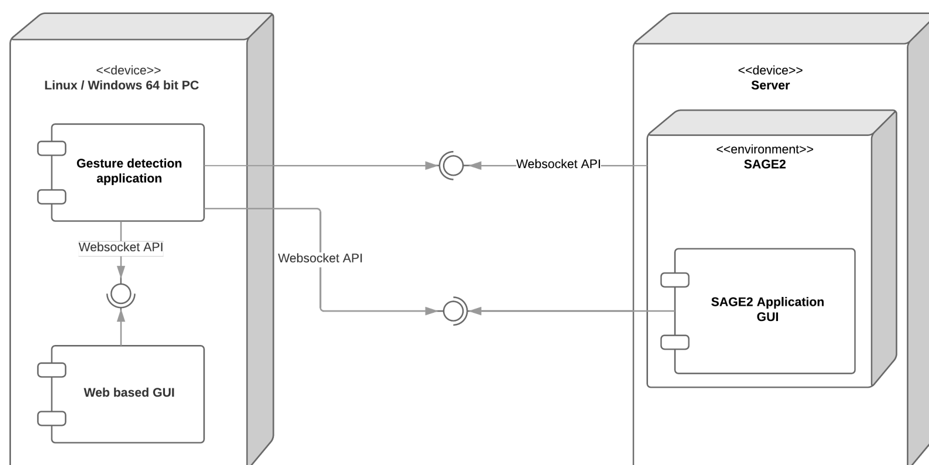


Figure 3.10: Deployment diagram

The deployement diagram in figure 3.10 shows the independent components abd their connections. I have decided to implement the connections using the websocket protocol[24], because it is a protocol created mostly for javascript based applications which need realtime communication. Communication in SAGE2 is also based mostly on websocket and the remote API in SAGE2 is only available using Websockets. The gesture application will provide an interface for the GUI and it will be available from any computer within the network. This will allow the user to run the GUI on any computer, which has access to the network.

3.4.2 External Libraries

The main purpose of this thesis is to implement an usable gesture control application for SAGE2 but the prototype application will require many functionalities which are not a subject of this project. For those functionalities I will use external libraries.

3.4.2.1 Skeleton Tracking & Kinect 2.0 Communication

There are two available options to manage the Kinect 2.0 input. One is the official Microsoft SDK[20] and second option are the open source libraries from the OpenKinect community[25] combined with NiTE2 library[21].

The application is required to run on the computers in SAGElab which mainly run on Linux operating system. Because of this requirement I cannot use the official SDK, because it is only supported on Windows operating systems. Therefore I will use the NiTE2 library in my implementation.

The NiTE2 library requires to use the C++ programming language. There are wrappers which allow me to use the library in different languages[26], but these wrappers would only require additional complications and I will only consider using them if I find any restriction which will not let me use C++ as my main implementation language.

3.4.2.2 Websocket Communication

As mentioned in previous section, I will use C++ as my main implementation language, therefore I will be looking for libraries also written in this language.

There are several libraries available for the websocket communication.

Qt Websockets[27] This library is a part of the Qt library, which is a very complex library. The websockets are well docummented and supported and support both client and server side of the protocol. The disadvantage of the library is, that it is dependent on the full Qt library, which is fairly big, about 2 gb of data.

easywsclient[28] This is a very lightweight library. The whole library is in a single cpp file. This library only supports the client side of the protocol.

Websocketpp[29] This library supports both client and server side. It also supports the encrypted version of the protocol. The only dependencies of this library are the asio library[30] and openssl[31] library for encrypted connection. The library requires C++11 compatible compiler[32].

Libwebsockets[33] This library supports both client and server side including the encrypted version. The encrypted version requires the openssl library or some of its alternatives. This library only supports linux operating systems.

I have decided to use the *Websocketpp* library. Mostly because this library has only few dependencies, while it supports all of the features I need and is multiplatform, which is the main advantage when compared to the *libwebsockets* library. The C++11 requirement is not very restrictive for my purpose, because usually only embedded devices do not have the C++11 support. I am not aiming to support these devices, since the Kinect 2.0 requires USB 3.0 and more CPU performance than most of these devices have.

I will need to use the openssl library to enable the encrypted communication. This is required because, the SAGE2 in sagelab runs on secure https protocol and web browsers do not allow insecure websocket connection, when the https protocol is used. Therefore the SAGE2 application GUI will need to use the secure Websocket connection.

3.4.2.3 JSON Support

In websocket communication, the data is most commonly transfered in JSON format[34]. Also the SAGE2 remote API send and receives data only in JSON format. To parse the JSON messages reliably, I have decided to use a library.

There is a large selection of libraries in C++[34]. My aim was to choose the most lightweight option, which will not require too much additional work when installing my final prototype. Based on feedback given on the library[35], I have decided to use SuperEasyJSON library[36]. This library only consist of two small files and can be included as source code.

3.4.2.4 Licences

All of the libraries I have selected use open source licenses, which allow me to use them and modify them in my project. The only exception is the NiTE2 skeleton tracking library, which is not open source but it is licensed as freeware, so I can use this library in my project, but I am not allowed to modify the library itself.

Chapter **4**

Implementation

This chapter describes the implementation of the prototype application. In this chapter, the main algorithms important for this particular application will be described. I will not describe commonly used programming patterns I used, for those are not the main topic of this thesis.

4.1 Code Style

When writing the code I will be using the *CamelCase*[37] naming convention. The main purpose of having a code style standard is to prevent inconsistency in the code. Which code style is the best is a very subjective question and in my opinion the main purpose is to keep the code consistent, no matter which exact code style the programmer chooses. I chose the *CamelCase*, because I am familiar with the style, which will make it easier for me to keep it consistent.

In the *CamelCase* naming, all variable, method, class and file names, when consist of more than one word, the words are not separated and every word begins with a capital letter. Variable and method names start with a lower case letter. Exception are constants which will be all capitals and individual word withing the name are separated by undescores. Here are few examples: ClassName, FileName.cpp, variableName, methodName, CONSTANT_NAME.

The code will be documented using doxygen comments[38]. From these comments the documentation is generated using tool Doxygen[39]. The tool generates documentation in HTML format.

4.2 Loading Configuration

Since the application will broadly use a JSON format for communication, I chose to use this format for the configuration file as well. It is not so common to use JSON format for configuration files within C++ application, but it is becoming the new standard for most of the applications running on web and

communicating over the network. Lastly I will already be using a library to work with JSON format strings, so this will simplify the implementation.

The configuration file will be loaded right after starting the *main* function. The configuration file will be loaded using a static class *Config* and its method *init*. The method looks for a file named *config.json* in the main folder of the application. If the config file is not found, default configuration is used.

4.3 Initialization & Main Loop

After loading the configuration, the main loop initializes the main parts of the application. First it runs the websocket server, which serves the user interface clients. This server i started in a separate thread. Then, the application connects to the SAGE2 server using the *SAGE2Controller* class. The hostname of the SAGE2 server is fetched from the configuration file or it is accepted as an argument, if the user adds the *-hostname* or *-h* switch with the hostname of the server when running the application.

The main loop of the application runs a *while* loop, until a stop is requested by the UI or the user presses a keyboard key. In the *while* loop, a frame is read using the NiTE library and its *UserTracker* class. The user tracker detects all the users in the frame. If the user is visible and calibrated, these are statuses provided by NiTE, the userdata of the user is passed to the gesture detection algorithms which are described in the next section.

4.4 Gesture Detection

There are 5 classes, where each recognizes a different type of gesture. Each of the classes can receive a user data retrieved from each frame by the NiTE library and from the user data it detects the gestures. The user data contains basic information about visibility of the user and position of each detected joint in the body. The NiTE library detects 15 different joints in the body, such as the head, torso, shoulders, elbows, hands, hips, kness, etc. Each of the joints contains the probably position of the joint in 3D dimensional space, and the confidence of this position. The confidence of the position tells the probability of the position being accurate. If the confidence is *0.5*, the position is 50 % accurate. Most of these gestures have several states. When a state of any gesture is detected, along with additional information about the gesture it is passed to a *GestureMapper* class which is described later in this chapter.

All of these classes inherit from a class called *GestureDetectionBase*. This class implements some common helping methods such as methods to extract individual joints from the user data. For example: *getLeftHand*, *getRightHand*, *getHead*. Also another common methods in this class are *isPositionConfident*, *isPositionExtraConfident*, these methods verify how confident is the position of the given joint according to the predefined constants.

4.4.1 ActiveDistanceDetector

This class is necessary for the presentation mode. It verifies whether the user is in the active distance. The distance is defined in configuration. To verify the distance, the algorithm takes position of the users head, if the position of the head is not detected confidentally enough, the algorithm takes the position of user's hips. If the position is within active distance, the *onActiveZoneEntered* event is triggered on the *GestureMapper*, *onActiveZoneExited* is triggered otherwise. If the position of the user is not confident enough, no event is triggered.

4.4.2 PointGestureDetector

This class handles all the gestures which involve pointing at screen. Since very often the hand covers the user's head to the Kinect sensor, the head position is calculated as an average of 10 last confident head positions, while testing, this improve the correct detection of the gesture, because the head was very often invisible, but In most cases, we can expect the head to be in the same position as in the previous frame. When the user's hand is fully stretched, the pan gesture is detected, when the hand is pointing at screen but not fully strechted, the point gesture is detected. The fully strechted arm is detected, when the distance of the hand from the shoulder is more than 30 cm. The length of the stretched arm is defined in configuration, but experimentally I have chosen 30 cm, because when trying larger number, the gesture was not being detected very often. The length of the arm is very inaccurately detected and in future solutions should be replaced by something more efficient. The accuracy problems are described in the *Calibration* section. Since the shoulder is very often covered by the rest of the arm, the position of the shoulder is saved in each frame and if the shoulder is not visible, the position from the last frame is used.

The data passed to the gesture mapper are the hand side, position of the user's head and direction of the pointing vector. The direction of the pointing vector is calculated as $handPosition - headPosition$. This information is then used to calculate the point on the screen, where the user points to. This is described later in this chapter.

One of the issues that came up after implementing this gesture was, that a slight movement of the hand results in a rather large movement of the cursor. So even when the user holds the hand steadily, the cursor still jitters. I tried dampening the jitter by using an average of last few positions of the hand. This reduced the jitter slightly, but it also caused the control to look slow. The response to the hand movement became more fluent but also very delayed. I tried reducing the number of last positions that were included in the avarage, but the delay was not present only when using less than 5 last positions and this amount didn't reduce the jitter noticeably. So I have omitted the dam-

pering in this prototype, and in future solutions a more advanced dampering has to be used.

4.4.3 RaiseHandGestureDetector

This class detects the hand raise gesture. When the user's hand is raised above the position of the user's head, start of the hand raise gesture is detected. When the hand is lowered below the head level again, hand raise gesture finish event is triggered. The start of the hand raise is only detected, when the hand is right above the head or above and behind the head. This prevents the hand raise from being detected, when the user points at the top of the screen.

4.4.4 VerticalHandSelectionDetector

This class is only for the presentation mode. It is used for the windows selection when switching windows in presentation mode. The algorithm uses two points, the bottom point, which is the position of the user's hips and the top point, which is the top of user's head. The detection algorithm checks the vertical position of a hand and returns the relative position between the bottom and the top point. The relative position is a number between 0 and 1, which is calculated as follows: $(handPosition.y - bottom.y)/(top.y - bottom.y)$. This number is passed to the gesture mapper.

4.4.5 WaveGestureDetector

This class detects the wave left and wave right gestures, which are then transformed to left and right arrow keys. The algorithm is based on a calculation of velocity of the user's hand in a specific direction. The algorithm keeps the recent points in a queue along with their timestamps. All points older than a configured time are disposed, default configuration is 1 second. The algorithm compares the distance of the last saved point and the first saved point in the queue. If the distance of these two points is more than the configured distance, default is 40 cm, the hand travelled this distance in less than is configured time and therefore reach the required speed to detect the wave gesture. By comparing the x coordinate of the first and last point, the algorithms detects whether the hand was moving from left to right or right to left and desired wave gesture event is triggered.

After the wave gesture is detected all of the currently remembered points are disposed to prevent multiple detection.

4.4.6 Calibration

In the design of the solution, I have suggested an user calibration which would improve the detection of the gestures. I have implemented this calibration

to improve the point gesture detection, but the calibration didn't work as expected. The problem is, that in the design, I suggested measuring the user's lengths in T-pose and then use these lengths while detecting the gestures. I tried using the length of an arm to detect when the arm is fully stretched, but it seems, that the distances of the joints differ in the different parts of the field of view. This is mainly caused, because when the user points towards the screen, the pointing hand covers other joints of the body and this causes positions of these joints to be inaccurate. To solve this issue, more investigation is required, therefore the implementation of the calibration will be omitted in this prototype.

4.5 SAGE2 Communication

SAGE2Controller is the class which handles communication with the SAGE2 server. In constructor the class receives the hostname and port of the SAGE2 server. The object connects to the SAGE2 server as a websocket client and let's the server know that the client wants to receive all of the available information about windows in the SAGE2 environment.

This objects keeps listening to websocket messages from the server in a background thread. It also keeps track of all the windows in SAGE2 and their positions, this enables this controller to have full control of the windows.

The public interface of this class offers methods to manipulate with size and position of the windows and also to use the pointer inside the SAGE2 applications. The public methods are then converted into JSON requests and sent to the SAGE2 server.

4.6 Gesture & Actions Mapping

The gesture and actions mapping is done in the *GestureMapper* class. This class receives events from the detector objects described earlier in this chapter and these events and current state of the object decide what action is performed. The states of this object represent the states of the whole gesture detection application which are described in the analysis. The state diagrams which describe the states of this object are in figures 3.5, 3.6, and 3.7.

One of the important functionalities of this class is calculating the point which the user points at using his or her hand. In figure 4.1, you can see the geometrical representation of the gesture. The information received by the gesture mapper is the *head position* and the *pointing vector*.

To calculate the intersection point, I am using the following equation:

$$head_point + t * pointing_vector = intersection_point$$

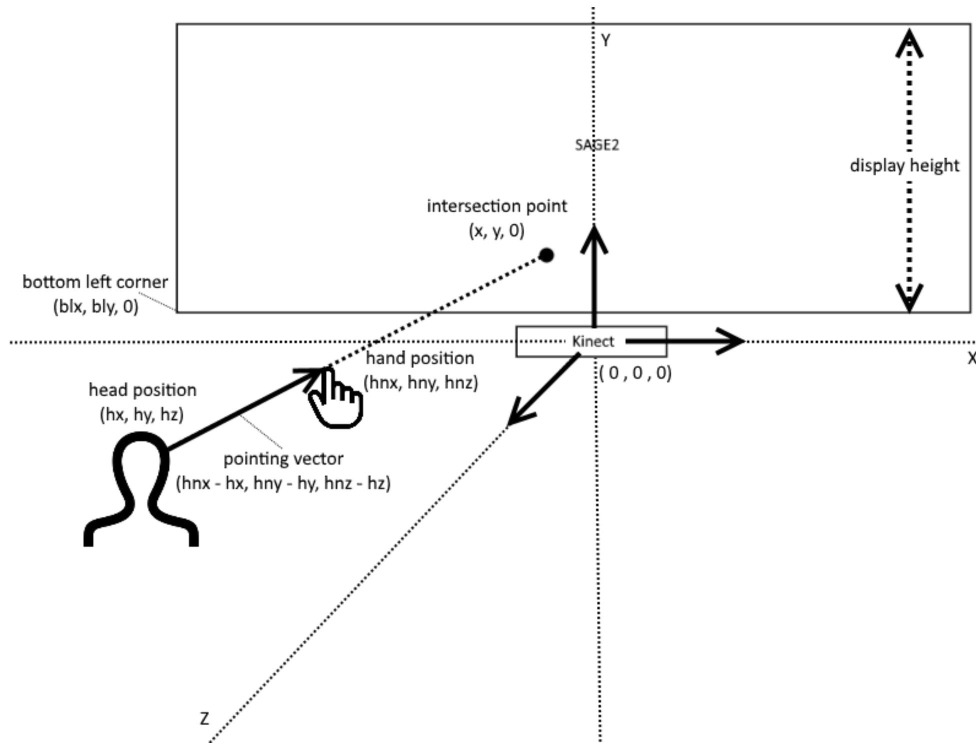


Figure 4.1: Point gesture to cursor position converstion

Where t is unknown. I know that the z component of intersection point equals 0. I can calculate t as follows:

$$t = \frac{intersection_point - head_point}{pointing_vector} =>$$

$$t = \frac{0 - hz}{hnz - hz}$$

When I have value fo t, I can easily calculate the x and y components of the intersection point:

$$x = hx + t * (hnx - hx),$$

$$y = hy + t * (hny - hy)$$

Now I have the position on the screen the user is pointing to, but I have the position in real world coordinates and I need to convert it to coordinates within the screen. In cofigurartion is set up the position and size of the display wall and from this information I can calculate the position within the screen. To calculate the point within the display wall I use following calculations:

$$display_point = intersection_point - bottom_left_corner$$

The problem is, that the coordinate system inside SAGE2 is reversed, the origin is in the top left corner and the y coordinate is positive bellow the origin, which is exactly the opposite from the Kinect coordinate system. So to reverse the y component of the display point I do following calculation:

$$display_point.y = height - display_point.y$$

Now we have the position on the display in centimeter and we need to convert it to pixels. I use resolution of the display wall to achieve that. The display resolution is fetched via SAGE2 api, when the SAGE2 controller is initialized. The converstion to pixel is done by following calculation:

$$display_point.y = resolution.y * display_point.y / display_height$$

$$display_point.x = resolution.x * display_point.x / display_width$$

Now we have the coordinates in pixels, which are sent to the SAGE2 as a position of the pointer cursor.

4.7 Graphical User Interface

The graphical user interface is fairly independent from the rest of the application logic. The first part of GUI is a part of the c++ application and is served by the *UIController* class. The second part of the GUI is either an HTML client in a webrowser or the SAGE2 application.

4.7.1 UIController

This class starts a websocket server which listens to incoming connections. The main loop of the websocket server is in a separate thread. The number of clients connected is not limited.

The websocket server works as a broadcast server. This means, that any message the is sent by the server is sent to all of the connected clients. The method offered in the public interface are these:

updateUser This method receives the user data and sends the individual joints of the user to the client. This allows the client to draw the skeleton of the user.

updateUserStatus This method receives the status of the detection of the user, such as *visible*, *invisible*, *calibrating*, *ready*, etc and sends it to the client.

updateHandStatus This methods receives whether a hand is currently active or not and which hand it is. An active hand is a hand, which is currently detected to be performing a gesture.

showSelectionItems This is a functionality used in the presentation mode. When user trigger a window selection, the list of available windows is sent to the client, so the user has some feedback during the selection.

selectItem This methods sends an index of the item which is currently being selected. The item is one of the items sent using the previous method.

selectionComplete This methods notifies the GUI, that the selection is finished.

Among these public methods, which are used for the communication from the application to the GUI, there is one method for receiving the incoming data. The method is called *getReceivedData* and as a parameter accepts a function name. The function name is a key which the user interface client sends to distinguish who is the receiver of the data. The objects, which expect some data from the client pass the key value to the method and the method returns string data which was received from the client under this keyword.

4.7.2 HTML & SAGE2 UI Client

Both versions of the UI client share the same main logic, which is written in javascript. The javascript object, which controls the UI is named *Kinect2forSAGE2*. This object has a start function, which draws the UI in the HTML element passed as a parameter and also creates a websocket and connects to the hostname and port passed in the argument of the start function.

The messages received through the websocket are in json format and have the following syntax:

```
{
    "f":"function name",
    "data":"received data"
}
```

In the *onmessage* event of the websocket, which is the event called when a message from the server is received, the *"f"* field is used to decide which function will process the incoming data. The expected function names are these:

- *updateUserData*
- *updateUserStatus*
- *updateHandStatus*
- *updateInteractionMode*
- *updateUserCount*

- *showSelectionItems*
- *selectItem*
- *selectionComplete*

The data sent in these messages is described in section 4.7.1.

The visual elements of the UI are created in the javascript code. The *start* function accepts an HTML element, which is the root element of the view. The prototype of the UI is shown in figure 4.2. The text elements are all simple html nodes and the skeleton is drawn using the canvas element and javascript drawing[40].

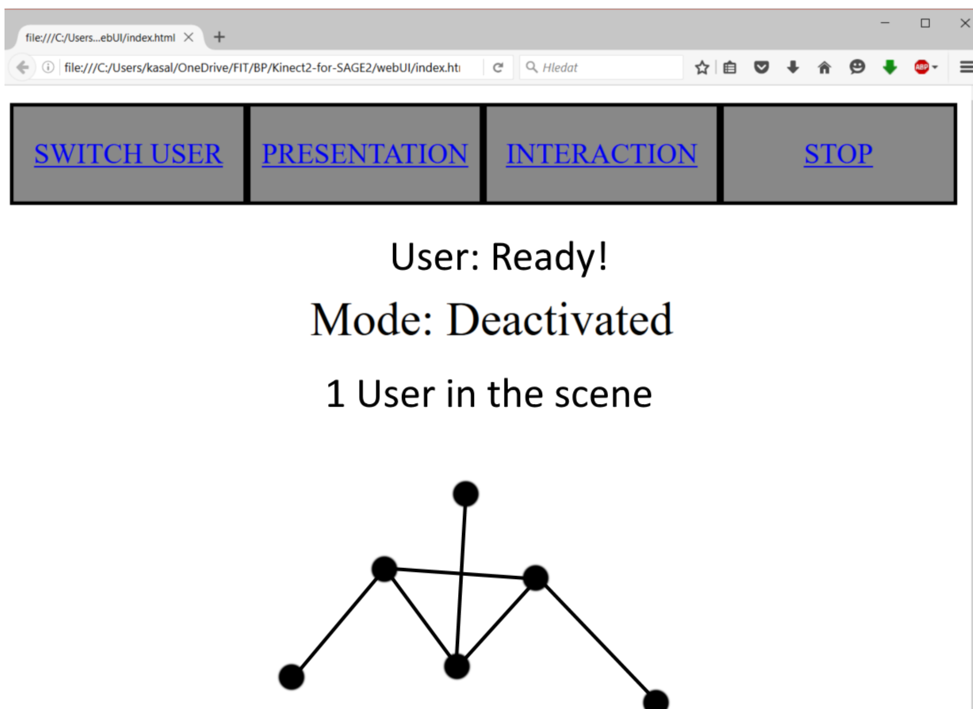


Figure 4.2: Web based graphical user interface

The buttons at the top of the view trigger functions which send a message to the server part described earlier in this section.

Chapter 5

Testing

Software testing is a standard part of the software development process. The regular software testing evaluates the software's performace and functionality. This type of testing was already being performed during the implementation to verify, that the functionality was implemented correctly.

For my prototype I will also conduct usability testing. Purpose of this type of tests is to determine, whether the software is easy to use for a regular user and if it performs well as a natural user interface[41].

For my usability testing, I will invite several students from Czech Technical University. None of them are familiar with my application so I will be able to analyze their performance as first time users. The testing group will consist of around 10 people.

5.1 Test Scenarios

The test scenarios are guidelines for the test subjects to follow during the testing. The scenarios are designed to require usage of all of the implemented gesture combinations, but at the same time the scenarios simulate a real life usage.

The test users will be given an user manual with description of all the implemented gestures and also they will be shown the individual gestures.

5.1.1 Interaction Mode

First tested scenario is in the interaction mode of the application. There will be 4 open windows in the SAGE2 environment: Google Maps, Movie Player, PDF Viewer, and SAGE2 UI of my application. These windows will be tiled next to each other. The initial configuration is shown in figure 5.1. The application starts as deactivated.

The user will be asked to perform following steps. Each step contains a description how to correctly perform the step. This description will not be

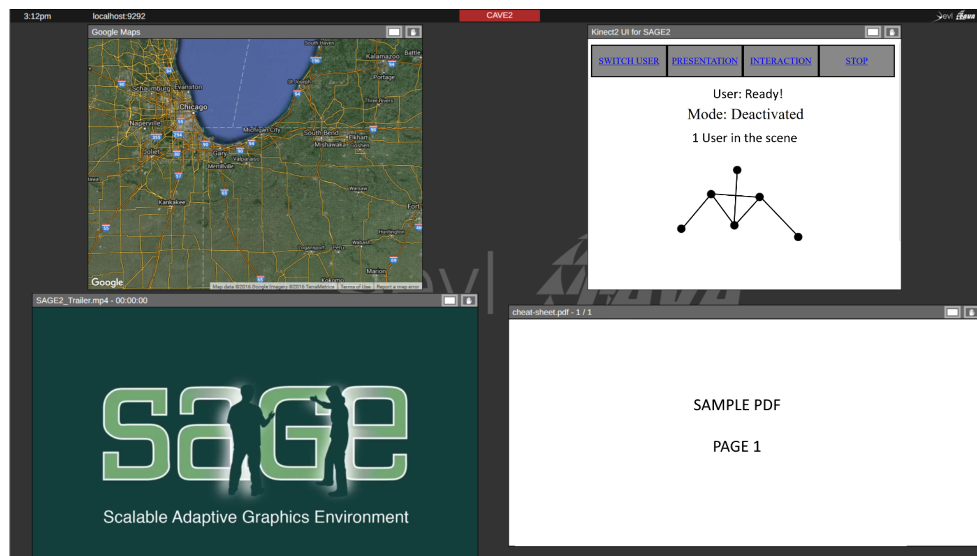


Figure 5.1: Initial setup for testing

available to the test user, the purpose of the description is, to track errors of the users by the moderator.

1. Activate the detection. Move and resize Google Maps application, to cover most of the screen.

   The correct result of this step is to activate the detection by raising user's hand, grabbing the Google Maps window, by pointing at it with a stretched hand and moving it towards the top of the screen, and resizing the window to the whole screen by pointing second hand. To the bottom of the screen.

2. Switch to Google Maps application interaction and zoom to Prague, so the city covers most of the window.

   The correct procedure for this step is to switch into the application by raising hand and pointing to the Google Maps window. Inside the application the user has to use the pan gesture - click in the map by stretching a hand and move with the hand to move with the map, then pull the hand back from the screen. By repeating this gesture the user can move the map to have Prague within the window. Then the user needs to use scroll gesture. This is done by pointing into the window by both hands, with the hands spread. When the map is zoomed to Prague, the user pulls the hand back from the screen.

3. Switch back to windows interaction, scale down Google Maps again and put them in the left top corner.

I expect the user to raise hand to switch back to windows interaction. Then grab the window by stretching an arm towards it, scaling down the window using the second hand and move it to the top left corner.

4. Switch to the Movie Player and start the video.

   To switch to the Movie Player application, the user has to raise his hand and point at the window of the Movie Player.

5. Switch to PDF Viewer and flip 5 pages forward.

   To swtich to the PDF Viewer, same as in previous step, the user has to raise one hand and point at the window of the application by the second hand. To flip 5 pages forward, the user needs to perform the wave gesture, wave one hand from right to left quickly. This gesture needs to be repeated correctly 5 times.

6. Deactivate the detection.

   When the application interaction is active, to deactivate the interaction, the user needs to raise one hand and put it down once to switch to windows interaction and then raise hand and put down again to deactivate the detection.

5.1.2 Presentation Mode

In the second scenario, the application will be in the presentation mode. The screen will be setup as shown in figure 5.2. At the begining the user will be outside the active distance of the sensor and will be asked to step out of the active distance after each step. The room will be set up in the presentation mode, so the user will have the feedback monitor against him available. In SAGE2, there will be these 3 applications prepared, PDF presentation, Video player and Google maps.

The users will be asked to performed the following steps:

1. Go 5 pages forward in the PDF Viewer.

   This step is performed by waving user's hand to the left 5 times.

2. Switch to Video player.

   To switch to video player, the user has to raise a hand, this will trigger the selection menu to appear, and in the selection menu the user switches between the items by positioning the hand up or down. The video player is the top item in the menu, so to select the top item, the user has to keep the hand in the top position for 3 seconds.

3. Play the video.

   The video is played by a single wave left gesture.

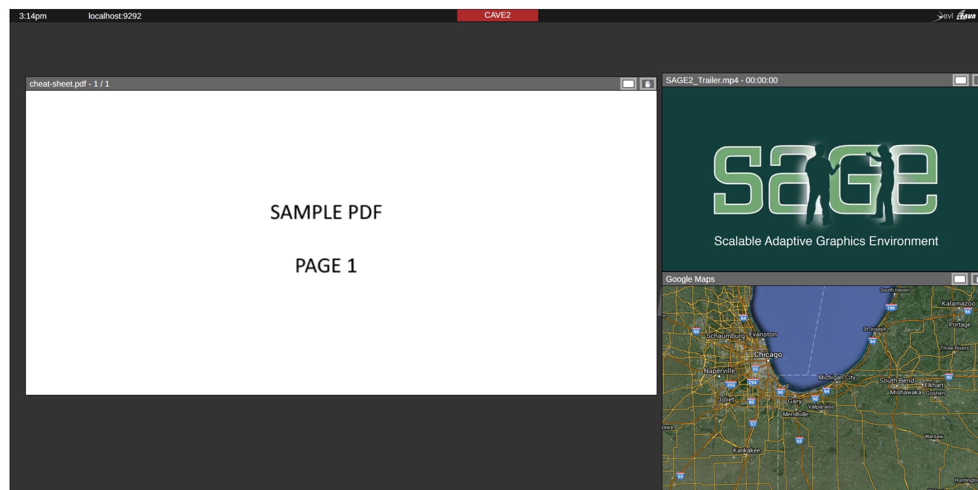


Figure 5.2: Initial setup for presentation mode testing

4. Switch back to the PDF Viewer.

   To switch back to PDF viewer the user has to perform exactly the same steps as in step 2 of this scenario.

5.2 Performance Evaluation

In usability testing, the results has to be objective and specific to provide a valuable feedback. For this reason, the evaluation has to convert the testing experiments into specific measurements.

First performance measurement will be the number of successfully finished steps in the test scenarios. When performing the test cases I will also count the number of errors during each step. An error occurs when user accidentally triggers a different action than intended or when the user performs a gesture, but this gesture is not recognized.

Last performance mesurement will be a survey. The first part of the survey will ask the users to rate each of the gesture usages from 0 to 5, where 0 is the worst and 5 is the best rating. The list of the actions I will ask the users to rate is in the following list. Along with the rating, I will encourage the users to write a short feedback under each of the actions, if they have any.

- Switching interaction modes
- Moving windows
- Resizing windows
- Moving inside application

- Zooming inside application
- Playing video
- Flipping pages in PDF
- Switching windows in presentation

In second part of the survey the user will be asked to rate the overall application and to provide a verbal feedback on the application.

5.3 Results

The testing was conducted with 6 different students. The experiments were anonymous, so I label the users by numbers from 1 to 6. All of the users were able to finish all of the steps of both scenarios.

In table 5.1, you can see the average rating of the individual actions along with the average number of errors in percent per each trial of performing the action, which I have counted while observing the users performing the gestures. I have divided the errors into these two groups:

Unrecognized The user performed the gesture, but it wasn't recognized.

False detection The user performed a gesture, but a different action than intended was triggered

Table 5.1: Gesture rating and error counts

Action	Rating (0 - 5)	Unrecognized %	False detection %
Switching interaction modes	4.16	14.3	9.5
Moving windows	3.83	11.9	21.4
Resizing windows	3.33	19.0	9.5
Moving inside application	2.33	11.9	40.4
Zooming inside application	1.5	47.6	35.7
Playing video	3.5	52.4	0.0
Flipping pages in PDF	3.16	47.6	4.8
Switching windows in presentation	4.33	14.3	0.0

From the results and also from observing the testing, we can see, that the windows interaction gestures performed pretty well, the verbal feedback from the surveys confirms this as well. The problematic gesture was the zoom

gesture inside the applications. All of the users expected this gesture to act more similar to how this gesture is done on a touch display. One of the issues is also that grabbing the windows and the content of application during the pan gesture is based on how stretched the hand is. Another problematic gesture was also the gesture for playing video and flipping pages in PDF viewer. The settings for the gesture require the movement to be too fast so many users had problems performing the gesture successfully. The rating of this gesture is interestingly not as bad as expected from the number of errors.

Chapter 6

Future Work

The results of this work are promising, and the testing provided a lot of valuable feedback for future work on this solution. The application prototype showed a potential and with additional work, a really useful solution can be created.

6.1 Gesture Improvements

The detection of the individual gestures is still not accurate enough. To make this system usable in real life scenarios, the detection must be improved. This section described the possible improvements, which came up from analyzing the test results and also during implementation.

6.1.1 More Kinect Devices

One of the main issues that cause inaccurate gesture detection is, that while performing the gesstures, the user covers the view of the Kinect sensor. For example during the point gesture, very often the users head and shoulder is hidden behind the user's arm.

This problem could be solved by utilizing more than one sensor. There could be another sensor capturing the scene from the top and from the side. All of the information from all three devices could be combined to retrieve more accurate data about the user.

6.1.2 Finger Based Gestures

This thesis focused solely on skeleton tracking and gestures detectable from the skeleton data. To detect more advanced gestures, the in further work, I should focus more on possibilities of adding fingers detection and combine skeleton based gestures with gestures performed using fingers.

One specific example, which was also mentioned in the analysis, is detection a clenched fist for a grab gesture. When the user wants to move a window, to grab the window, the user would clench the fist, which will indicate the grab. This gesture would feel very natural. To be able to detect this gesture, a more advance image recognition is required.

6.1.3 Combination With Other Devices

To be able to detect more advanced gestures, another possibility is also to combine the Microsoft Kinect with another device. For instance, SAGElab owns a device called Myo[42], which is specifically designed to recognize palm and finger gestures. If the gestures recognized by this device were combined with the skeleton based gestures, it would strongly improve the user experience.

6.1.4 Utilize Machine Learning Techniques

Computer vision very often utilizes machine learning algorithms. This work didn't consider using those technologies, but there is a lot of potential in using machine learning with skeleton tracking. Many of the gestures would certainly perform much better with the use of machine learning, than they do using regular algorithms. Also the advantage of such solution would be, that the solution could constantly be learning and improving.

6.2 Additional Functionality

The gesture detection improvements should be the main focus of the future work, because the detection accuracy influences the user experience the most, but there are also ideas for future improvements of the gestures and to add more functionality to the gesture control system. These are the suggestions.

6.2.0.1 Window Maximize & Minimize

To make it easier to show some window in full screen, there will be gesture to maximize it to the whole screen. This will be done by using the window positioning gesture. When the user drags the window above the top of the screen, the window will be streched to the whole screen. The previous size will be remembered. When the maximized window is dragged from the top of the screen, its previous size is restored. This gesture is very common in the Windows operating systems.

6.2.0.2 Closing Applications

This gesture will also use the windows positioning gesture. When a window is dragged to the center of the bottom of the screen, it will close the application. to provide some feedback to the user, when the window is at the poition to

be closed, it will get temporarily resized to a minimal size. In this position, if the grab gesture is released, the window will be closed. If the user pulls the window up again, it will restore its previous size and the close action will be cancelled.

Conclusion

The goal of this bachelor's thesis was to research current technologies which use depth sensors and gesture detection, and to use the collected knowledge to design a gesture control solution for SAGE2 using Microsoft Kinect 2.0. The focus while designing the solution was to create a natural user interface with the use of hand gesture detection. To verify the usability of the suggested solution, I created an application prototype, implemented according to the before mentioned design and performed usability testing on the prototype to receive feedback from practical usage.

I have performed a very thorough analysis of the possible gestures and of their possible usages. The analysis also contain testing of the individual gestures to obtain objective and accurate data about the specific gestures. This helped me estimate the performance of the gestures more accurately and it helped me design my solution using the best performing ones.

The solution designed in this thesis utilized many complex ideas and described the functionalities the resulting application should contain in detail. To test the ideas and estimate the performance of the resulting solution, I have developed a prototype application which was implemented according to the design and I have conducted an usability testing on this application prototype. The testing showed, that the solution shows a great potential and the feedback from the users confirmed, that the solution can be a valuable tool for the SAGElab users. At the same time, the testing has provided a lot of valuable feedback on where to go next with the solution and which ideas did not work in reality as expected when they were designed. Most of the problems were caused by the inaccuracies in the gesture detection.

The prototype application is publicly available and there already is a plan to continue the development of the gesture driven control in SAGElab. There are many possible augmentations that occured during the implementation and testing which were not considered in the design and I believe that using the knowledge gained during work on this thesis will allow the next development to improve the solution and create a complete and very usable solution.

Bibliography

[1] Sedláček, L. *SAGElab - Bezdotykové ovládání.* Bachelor's thesis, Czech Technical University, Faculty of Information Technology, Prague, 2015.

[2] *Reddit Xbox Dev talks Kinect 2* [online] . Available from: http://www.ign.com/boards/threads/reddit-xbox-dev-talks-kinect-2.453298003/

[3] *Depth Camera* [online] . Available from: https://msdn.microsoft.com/en-us/library/hh438997.aspx

[4] *How Does The Kinect 2 Compare To The Kinect 1* [online] . Available from: http://zugara.com/how-does-the-kinect-2-compare-to-the-kinect-1

[5] *SAGElab - Síťová a multimediální laboratoř* [online]. Prague: CESNET, FIT a FEL ČVUT . Available from: http://sagelab.cesnet.cz/

[6] *Fakulta informačních technologií ČVUT* [online]. Prague . Available from: http://fit.cvut.cz/

[7] *SAGE^{TM}* [online] . Available from: http://sage.sagecommons.org/

[8] *Introduction - SAGE2* [online] . Available from: http://sage2.sagecommons.org/project/introduction/

[9] *Microsoft Kinect pro Xbox One* [online]. Microsoft . Available from: http://www.microsoftstore.com/store/msusa/en_US/pdp/Kinect-for-Windows-v2-Sensor/productID.298810500

[10] *EVL - Electronic visualization laboratory* [online] . Available from: https://www.evl.uic.edu/

[11] *The University of Chicago* [online] . Available from: http://www.uchicago.edu/

[12] *Official Kinect 2 0 Gestures Tutorial Xbox One* [online] . Available from: https://www.youtube.com/watch?v=P4naZ58wIto

[13] *Hand gestures to control Windows mouse cursor – Kinect for Windows* [online] . Available from: https://blogs.msdn.microsoft.com/msgulfcommunity/2013/05/16/hand-gestures-to-control-windows-mouse-cursor-kinect-for-windows/

[14] *Kinect controls Windows 7- WIN&I* [online] . Available from: https://www.youtube.com/watch?v=o4U1pzVf9hY

[15] *Gestigon* [online] . Available from: http://www.gestigon.com/consumer-electronics.html

[16] *Gesture control of TV graphics (CZ)* [online] . Available from: https://www.youtube.com/watch?feature=player_embedded&v=b20pTYa14kM

[17] *Nonfunctional Requirements* [online] . Available from: http://www.scaledagileframework.com/nonfunctional-requirements/

[18] *Functional Requirements* [online] . Available from: http://www.ofnisystems.com/services/validation/functional-requirements/

[19] Li, L.: *Time-of-Flight Camera – An Introduction* [online]. Dallas, Texas, USA: Texas Instruments, 2014 . Available from: http://www.ti.com/lit/wp/sloa190b/sloa190b.pdf

[20] *Kinect for Windows SDK* [online] . Available from: https://msdn.microsoft.com/en-us/library/dn799271.aspx

[21] *NiTE 2* [online] . Available from: http://www.openni.ru/files/nite/index.html

[22] *Javascript - Mozilla Development Network* [online] . Available from: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[23] *SAGE2 Application API* [online] . Available from: https://bitbucket.org/sage2/sage2/wiki/SAGE2ApplicationAPI

[24] *Websockets - Mozilla Development Network* [online] . Available from: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

[25] *OpenKinect* [online] . Available from: https://openkinect.org/wiki/Main_Page

[26] *Software - OpenNI* [online] . Available from: www.openni.ru/software/index.html?cat_slug=file-cat2.html

[27] *Qt Websockets* [online] . Available from: http://doc.qt.io/qt-5/qtwebsockets-index.html

[28] *Easywsclient* [online] . Available from: https://github.com/dhbaird/easywsclient

[29] *WebSocket++ - Zaphoyd Studios* [online] . Available from: http://www.zaphoyd.com/websocketpp

[30] *asio C++ library* [online] . Available from: http://think-async.com/

[31] *OpenSSL* [online] . Available from: https://www.openssl.org/

[32] *C++11 Overview* [online] . Available from: https://isocpp.org/wiki/faq/cpp11

[33] *libwebsockets* [online] . Available from: https://libwebsockets.org/

[34] *ECMAScript 2015* [online]. ECMA International . Available from: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

[35] *SuperEasyJSON Reviews* [online] . Available from: https://sourceforge.net/projects/supereasyjson/reviews

[36] *SuperEasyJSON* [online] . Available from: https://sourceforge.net/projects/supereasyjson/

[37] *CamelCase* [online] . Available from: http://c2.com/cgi/wiki?CamelCase

[38] *Doxygen Manual: Documenting the code* [online] . Available from: https://www.stack.nl/~dimitri/doxygen/manual/docblocks.html

[39] *Doxygen* [online] . Available from: http://www.stack.nl/~dimitri/doxygen/

[40] *HTML5 Canvas* [online] . Available from: http://www.w3schools.com/html/html5_canvas.asp

[41] *NUI (Natural User Interface) Definition* [online] . Available from: http://techterms.com/definition/nui

[42] *Myo Gesture Control Armband* [online] . Available from: https://www.myo.com/

Appendix A

Acronyms

GUI Graphical user interface

UI User Interface

JSON JavaScript Object Notation

SAGE Scalable Amplified Group Environment

Appendix B

Contents of enclosed CD

```
readme.txt ......................... the file with CD contents description
exe ...................................... the directory with executables
  readme.txt .................. the file with description of the executables
  run.bat ..................... script to run the application on Windows
  run.sh ........................ script to run the application on Linux
  user_manual.pdf ..................... user manual for the application
  index.html ............................... Web based user interface
  bin ......................................... binary files and libraries
  js .................... javascript logic for the web based user interface
src ........................................ the directory of source codes
  Kinect2-for-SAGE2 ...................... folder with the source files
    Doxyfile ..................... file with configuration for Doxygen
    install.sh .......... script to install OpenNI and NiTE2 libraries
    Makefile ...................... makefile with rules for compilation
    readme.txt ................. file with description and instructions
    run.bat ......... script to run the generated binaries on Windows
    run.sh ............. script to run the generated binaries on Linux
    bin .............................. folder for the generated binaries
    lib ................................ folder with the used libraries
    source ............................ folder with source code files
    UI ..................... folder with user interface implementations
      Kinect2UI ........... SAGE2 application for the user interface
      webUI .............................. web based user interface
  thesis ............. the directory of LaTeX source codes of the thesis
text ....................................... the thesis text directory
  BP_Tomáš_Kasalický.pdf ............. the thesis text in PDF format
```