TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Automaatikainstituut

ISS40LT

Martin Jaan Leesment 123826

KINECTI BAASIL TÖÖTAV TARK RAKENDUS ETTEKANNETE TEGEMISEKS

Bakalaureusetöö

Juhendaja: Eduard Petlenkov
Dotsent

Tallinn 2015

Autorideklaratsioon

Käesolevaga kinnitian, et esitatud töö „KINECTI BAASIL TÖÖTAV TARK RAKENDUS ETTEKANNETE TEGEMISEKS“ on minu isikliku töö tulemus. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikad ja mujalt pärinevad andmed on viidatud. Käesolevat tööd ei ole varem esitatud kusagil mujal.

Kuupäev:

Autor: Allkiri:

Declaration

Herewith I declare that this thesis is based on my own work. All ideas, major views and data from different sources by other authors are used only with a reference to the source. The thesis has not been submitted for any degree or examination in any other university.

Date:

Author: Signature:

Kinectil põhinev tark rakendus ettekannete tegemiseks

Annotatsioon

Antud töö eesmärgiks on luua Microsoft Kinecti baasil töötav programm, mille abil oleks võimalik juhtida esitlustarkvara kasutades liigutusi.

Rakendus on kirjutatud C++ programmeerimiskeeles Windows 7 operatsioonisüsteemiga arvutis. Suhtlus arvuti ja kasuta vahel toimub läbi naturaalse kasutusliidese. Lisaks on loodud ka lihtne graafiline liides.

Töö on kirjutatud inglise keeles ja sisaldab 34 lehekülge, 7 peatükki, 5 pilti.

Kinect based smart application for making presentations

Annotation

The objective of this work was to create a Microsoft Kinect based application for making presentations.

The application is written in C++ and works under the Windows 7 operating system. The communication with the computer is done through a natural user interface. A simple graphical user interface has also been created.

The thesis is written in English and contains 34 pages of text, 7 chapters, and 5 figures.

Table of Contents

List of illustrations

1. Introduction

The user interface history begins with the batch era. Users had to accommodate computers rather than the other way around, user interfaces were considered something of an overhead. Batch interfaces were followed with command-line interfaces. These introduced textual commands in a specialized vocabulary, users had to remember various commands. Then soon graphical user interfaces came to life. The next logical step in user interface design is towards something even more intuitive – a concept called natural user interfaces (NUI).

A natural user interface is effectively invisible – users interact with the computer naturally using their gestures and skeleton motions. It is believed to be the bridge between the physical world we exist in and the virtual reality we create, a completely new way of interacting with arts and also a profitable business opportunity for individuals and companies. Natural user interface applications bring us to augmented reality which enables users to feel as if they appear and live in a nonexistent world [1].

Microsoft Kinect introduced the notion of motion sensing to a wider audience. Kinect software development kit (SDK) [2] made it one of the first publicly available options for implementing NUI applications. In the beginning it gained its popularity through games, but soon after more research went into other more practical possibilites for everyday uses.

This thesis presents one such practical solution in an academic environment. A software implementaion that, with the help of Kinect, allows a user to make a handsfree computer presentation through a natural user interface by using gestures.

2. System description

The system consists of Microsoft Kinect and software written in C++ programming language. The software application runs on a personal computer with Windows 7 or Windows 8 operating system installed. Software was developed using Microsoft Visual Studio [3], Kinect SDK and some additional free-source libraries. All of these parts will be introduced in the following section.

2.1. Microsoft Kinect

Microsoft Kinect is a set of motion sensing input devices launched by Microsoft in November 2010 for Xbox 360, Xbox One video game consoles and personal computers running Windows operating system [4]. It captures a stream of colored pixels with data about the depth of each pixel. It also contains a four-element, linear microphone array that enables the user to capture positioned audio data.
The device allows users to interact with their console/computer through a natural user interface using gestures and/or spoken commands.

Kinect sensor is a horizontal bar connected to a small base with a motorized pivot and is designed to be positioned lengthwise above or below the video display.

It consists of an infrared admitter and receiver, a color (RGB) camera with a depth sensor, and a microphone array [5]. There is also a tilt motor that allows altering camera viewing angles. By using these components and a special microchip, Kinect can be used for facial and voice recognition, and tracking and identifying the movement of objects and individuals in three dimensions.

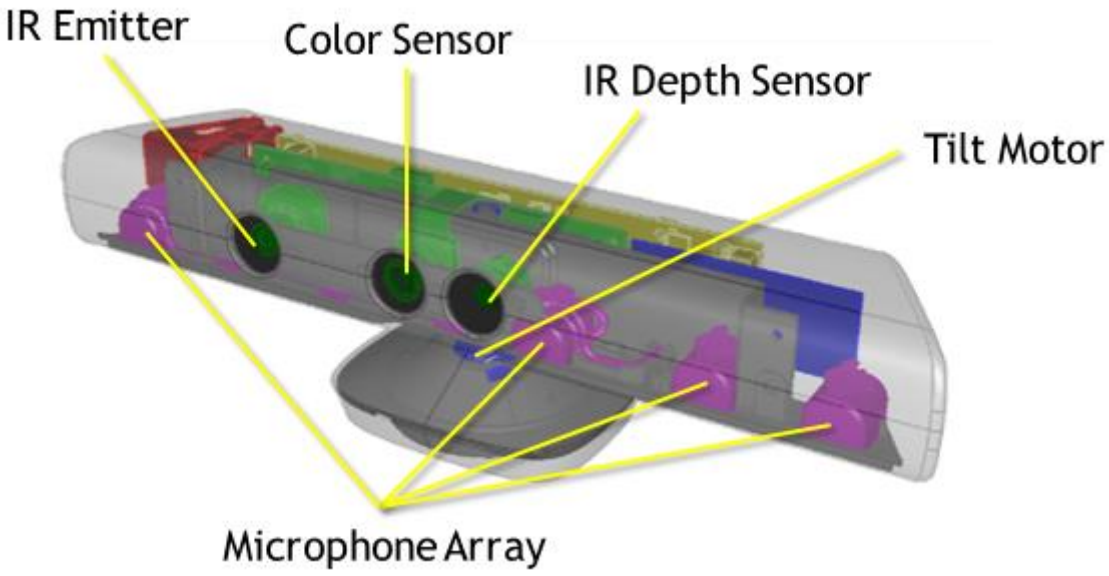

Figure 1. Kinect composition [5]

The devices communicates with a computer through a standard USB 2.0 port. The data processing is the responsibility of the Kinect drivers installed on the computer. Currently the drivers are supported on Windows 7 and Windows 8 operating systems (soon on Windows 10). There is also an open community OpenKinect [6] that works on creating drivers for other personal computer operating systems like Linux and Mac. The operating system should run on a 32-bit (x86) or 64-bit (x64) processor. Other requirements include two gigabytes of random access memory (RAM), a graphics card that supports DirectX 9.0c, and a dual-core 2.66GHz or faster processor [7].

2.2. Kinect SDK

Kinect SDK was released in spring 2011. It provides the tools and APIs, both native and managed, that are necessary for developing Kinect-enabled applications for Microsoft Windows.

The SDK install includes the NUI API and the necessary drivers to integrate the Kinect sensor with Microsoft Windows. Besides the necessary drivers and APIs, the install includes documentation, as well as the toolkit including resources and samples to help developers get started [8]. Currently it supports developing applications with C++, C#, or Visual Basic by using Microsoft Visual Studio integrated development environment (IDE) version 2010 or newer.

Developers using the SDK have access to raw sensor streams (color and depth), take advantage of skeletal tracking, and use advanced audio capabilities [9]. The first two, sensor streams and skeletal tracking, have been made use of in the thesis work at hand.

Users are recognized when they stand in front of the sensor with their head and upper body visible to the sensor. Tracking is automatic - no specific pose or calibration action needs to be taken for a user to be tracked [9]. When a person passes in front of the sensor, the skeleton model is created by the NUI library using depth data about the distance from background objects. Kinect can recognize six people and successfully track two them. For recognizing movement and specific gestures, a special skeleton model is used:

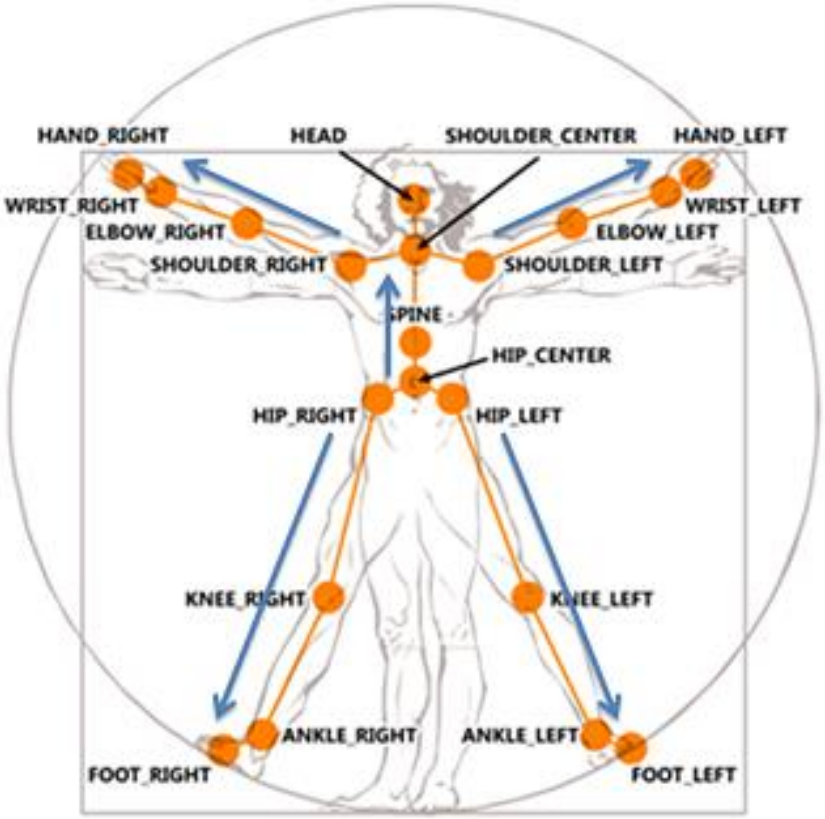

Figure 2.Kinect skeleton model [10]

The coordinate system used is a simple cartesian model with values returned in meters [11]:

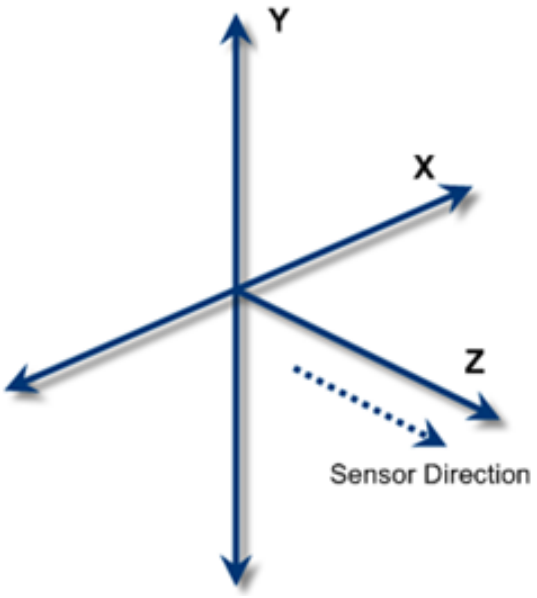

Figure 3. Kinect coordinate system [11]

Moving towards the camera decreases the z-value, while moving away increases it. The Y-coordinate increase upwards and decreases downwards. The X-coordinate value decreases when moving to Your right.

Kinect SDK supports different resolutions for the color and depth camera. The program developed uses 640x480 (maximum for depth camera, color camera also has a 1280x960 option) for both of them.

The NUI application programming interface (API) uses the depth stream to detect humans moving in front of the sensor.

2.3. Visual Studio

Microsoft Visual Studio is a proprietary IDE from Microsoft that allows users to create multi-platform applications. It is available as a free download for sudents [3].

Visual Studio includes a code editor as well as code refactoring, an integrated debugger that works as a source-level debugger and a machine-level debugger, and a source-control system [12]. Besides having access to state-of-the-art tools and services for creating applications,

developers can use microsoft developer network (MSDN) for accessing specification information, tutorials and getting guidance and support from specialists. Besides creating console application, Visual Studio also allows users to create graphical user interface (GUI) applications (the graphical user interface possibilities provided by the IDE were not utilized in this thesis work due to the necessity of drawing OpenGL graphics).

Visual Studio supports a variety of programming languages including C, VB.NET, C#, Python, Ruby and of course C++ in which the current program was written.
Visual Studio also automatically integrates the OpenGL library with every Visual Studio project, a feature very helpful in developing the program described in this thesis.

The current program was written in Visual Studio 2013.

3. Development

The development process started from initializing Kinect from C++ code written in Visual Studio to implementing a graphical user interface that would display a live image returned by the Kinect video streams. In the following section I describe the different parts used in the development process.

3.1. Augmented Reality with Kinect

The first steps were learning how to use Kinect and its SDK together with OpenGL. A book from Rui Wang [1] was used to get started. It provides guidance for developing Kinect-enabled applications with C/C++ using Kinect SDK, as well as introducing the FreeGLUT library [13] for OpenGL support. It helped to understand Kinect initialization, color and depth image streaming, and skeleton motion and face tracking. The basic idea on which the current thesis application relies on gesture recognition was also introduced.

Besides acquiring the methods on how to implement gesture recognition and ideas on structuring a FreeGLUT program, the book included a helpful header file (*GLUtilities.h* [14]) that includes some OpenGL specific drawing functions that are utilized in this work to paint graphics.

3.2. Kinect SDK

Kinect SDK functions are easily recognized with the prefix *NUI*. They provide access to information collected by Kinect sensors, for example how many devices are connected, the depth and color data, camera angle.

Kinect SDK functions were used to initialize and connect to Kinect as well as terminating the connection safely when exiting the program. All the image and depth pixel data and skeleton information that make live image and gesture recognition possible are acquired through the help of functions provided by this development kit.

Skeleton tracking is essential for this kind of program. In fact, if no graphics are necessary to be displayed, Kinect SDK solely suffices for making a simple program that reacts to different gestures. The parts of the body and their positions that are of integral importance in this work are spine, shoulders, hands, elbows, and head. Their coordinates are used to limit gesture area, track user movement and differentiate between gestures.

There are also special requirements when choosing the position for Kinect. According to MSDN [15], it is best if Kinect is placed relatively high on a stable surface [16]. It was discoverd during usage, that when placed too low, the application has trouble recognizing gestures.
For the program at hand, the viewing area should be from a little bit down from the waist to the top of the head. The waist down requirement comes from the fact that when lowered, hands extend a little further down from the waist. The program tracks hands, and when toggling between fullscreen, it uses the position of the hands to restart the movement. Hands need to be lowered for a new toggle event to occur. The top of the head requirement is important for every application that uses Kinect, because of face tracking [17].

When deciding on the location of the sensor, light is also an important factor. In a dark room, Kinect has difficulties recognizing movements and especially tracking faces. This means that at least the position where the Kinect and user are situated should be well-lit. This can prove to be problematic for the kind of program at hand – presentations can potentially be made in a dark room.

Another important factor is user positioning. The interacting user should not stand too close to the camera. The Kinect depth sensor range is from 800mm to a maximum of 4000mm. The Kinect for Windows Hardware can however be switched to Near Mode which provides a range of 500mm to 3000mm instead of the default range [11]. This possibility has not been implemented in the code.

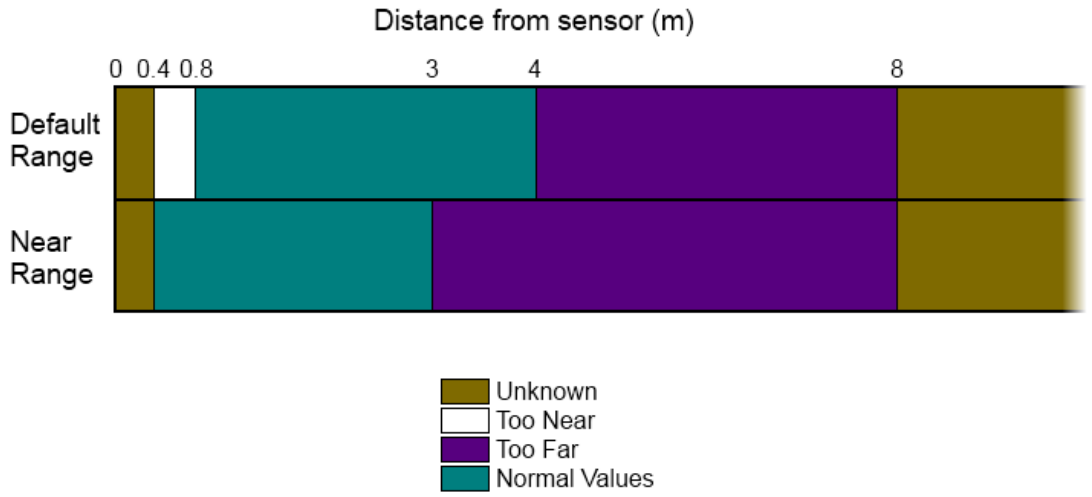

Figure 4. Kinect distance range [11]

Also, it is worth mentioning, that if there are more than one active Kinect device present, they should not be placed face-to-face as there may be infrared interference [18].

3.3. FreeGLUT

FreeGLUT is an open-source library that serves as an alternative to an ageing GLUT library (last version was released in 1998) allowing developers to create and manage OpenGL contexts with just a few functions and readable callbacks. Users can create and manage windows containing OpenGL contexts on a wide range of platforms [13]. It is also possible to read the mouse, keyboard and joystick functions (the latter features are not used in the application made with this thesis). For the work at hand the library was used to draw the graphics from Kinect sensor pixel information and keep the program constantly working thanks to the event processing loop.

GLUT uses callback functions. Some essential callbacks that are used in this application include (descriptions are taken from the OpenGL project page [19]):

- *glutReshapeFunc* – sets the reshape callback for the current window. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established.
- *glutDisplayFunc* – sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the display

callback for the window is called. GLUT determines when the display callback should be triggered based on the window's redisplay state. The redisplay state for a window can be either set explicitly by calling *glutPostRedisplay* (used with this application) or implicitly as the result of window damage reported by the window system.

- *glutIdleFunc* – sets the global idle callback so a GLUT program can perform background processing tasks or continuous animation when window system events are not being received. If enabled, the idle callback is continuously called when events are not being received. This is used to get live image from Kinect video streams.
- *glutMainLoop* - GLUT uses an event processing loop function that runs constantly during the lifetime of a program. This is also the callback calling entity. The routine should be called at most once in a GLUT program. This routine will never return.

3.4. GLUI

After the main functionality (changing slides) was achieved, the work continued with finding some suitable library for implementing a graphical user interface to show the live image from the Kinect camera. Different solutions were explored (wxWidgets [20], Ogre [21]), but in the end GLUI was chosen because of its close relationship with GLUT [22].

GLUI is a C++ specific user interface library based on the GLUT (and hence fully compatible with FreeGLUT) toolkit [13, 22]. It provides graphical user interface possibilities like building controls such as as buttons, checkboxes, radio buttons, and spinners [23] that are not possible to create without a lot of extra work and hacks in freeglut. For a commercial software product, it isn't probably the best GUI toolkit to use compared to the likes of wxWidgets, or Qt [24], but it has a small footprint and for use in an educational and experimental environment it is more than enough.

Like mentioned before, GLUI has a close relationship with GLUT, this is also the main reason for choosing it. GLUI made it possible to continue from the FreeGLUT and Kinect SDK solution that was created before work towards developing a GUI had even begun. Had some other library been chosen, a lot of rework would have been necessary.

3.5. Microsoft Developer Network

MSDN is a Microsoft commercial knowledge base that provides helpful resources and guidance for developers and testers interested in the operating system or in various software products from Microsoft [15, 25]. The site is freely accessible to everyone.

The main functionality of the program relies on the communication between the operating system and the application. After a gesture is recognized, the program responds by sending an appropriate keyboard event (or a combination of them) to the operating system. For implementing this behavior, MSDN provided crucial information.

From this site, it was possible to gather information on how to communicate with a Windows operating system [26]. Additionally it had an entire Kinect specific section that provided code samples and guides [8], plus a lot of helpful information about the device. All of the functions for using the tilt motor were read about from this site [27].

It should be mentioned that this Microsoft based approach makes the program incompatible with other operating systems.

3.6. Functions guide

The program consists of a header file and two source files: the main program with the GUI design (*main.cpp*) and a header for Kinect and GLUT functions (*KinectHeader.h*) with an associated source file (*KinectFunctions.cpp*). A brief description about the functions created follows:

KinectHeader.h:

- *initializeKinect* – the function that connects with Kinect and sets the necessary settings for tracking and video streams.
- *destroyKinect* – the function that safely closes the connection with Kinect.
- *initializeFaceTracker* – for initializing face tracking.
- *destroyFaceTracker* – for safely terminating face tracking.
- *guessgesture* – the function that uses hand positions to identify a gesture.

- *gestureHandler* – the function that is responsible for communicating with the Windows operating system and making a decision based on global boolean values when and what kind of input signals to send.
- *sendFullScreenEvent* – used to send an appropriate combination of keyboard input signals for switching the presented file to full screen mode.
- *updateSkeletonData* – used for getting data about the positions of different joints and body parts at any given moment.
- *quitProgram* – a function for safely exiting the program after a corresponding gesture has been recognized.
- *checkGestureRestart* – a helper function that is used to prevent a gesture to be sent more than once during a single gesture (necessary due to the fact that sometimes movement is processed slower than the callback calling mechanism in GLUT works).
- *myIdleFunc* – for getting the crucial information from Kinect. Registered as a GLUT idle function so it is called constantly when *glutmainloop* is not receiving any events. It is also responsible for calling *updateSkeleonData* and *updateImageFrame*.
- *updateImageFrame* – constantly (called from *myIdleFunc*) getting data from sensors about color and depth and saving these values.
- *render* – the callback for *glutDisplayFunc*. Used to paint the picture with the data got from Kinect sensors. Draws OpenGL graphics onto a GLUT window.
- *initKinectAndTexture* – function for calling *initializeKinect* from the main source file. Necessary because the GUI part is in a different file, and if only one of the functions would have been used, then it would have been necessary to share the graphical user interface parts with *KinectFunctions.cpp*.
- *setTheTilt* – created to prevent sharing Kinect context interface between multiple source files.

Main source file:

- *angle_cb* – function used to control the Kinect tilt motor. Enables the user to set the viewing angle through the graphical user interface.
- *returnCurrentAngle* – function for getting the current angle of Kinect and displaying it on the GUI.
- *setPreseExtension* – function that sets a global variable for enabling the program to choose between the correct keyboard inputs that lead to fullscreen mode (for all the supported presentation file formats, the fullscreen combinations are different)

- *connectToKinect* – the function that calls *initKinectAndTexture* from the second source file and display status information about the connection.

4. Application

4.1. Gesture recognition

Gesture recognition is the basis of the functionality provided with this work, so I think this would be the best place to start describing the application.

Gesture recognition is done in a way that was encountered the most in Kinect-enabled programs that I looked into – positions of different bodyparts relative to each other. Same approach has also been used in a lot of the works referred to in the „Similar Existing Solutions“ topic below.

For a gesture to be recognized, the user needs to stand in front of the camera so that Kinect can identify the skeleton and start tracking it. It is also necessary for the user to face the camera, this approach was chosen so that the program would not react to gestures when the presenter is facing the crowd and possibly giving an emphatic speech. For example if the Kinect is placed to the left of the user, but he or she is facing in another direction, then the gestures can not be recognized. This minimizes the chance of gestures accidentally being reacted to by the program.

Gestures recognized in this program:

- swipe towards the body with your left hand – move to previous slide
- swipe towards the body with your right hand – move to next slide
- raise both hands above your shoulders – toggle between fullscreen mode
- cross your arms in front of you – exit the program

Note that it is also possible to change the dominant hand – the user can choose that the left hand will move to the next slide, and right to the previous. Default is the other way around.

All gesture recognition is based on the position of the hands during and after a movement. For example, a swipe away from the body isn’t recognized as a gesture, because the initial and final positions do not satisfy the conditions of a slide changing swipe. Additionally there is a valid gesture area – the swipe must happen somewhere between the waist and shoulder positions.

Besides the correct area, speed plays also part in a swipe recognition process - a very slow or too quick swipe will not be recognized.
There is also the matter of longevity with toggle and exit gestures. Holding your hands up for only a couple of milliseconds is not recognized as a toggle event. Also, another toggle event will not be created until the user has not let his or her hands down for a moment. All of these scenarios are optimizable and source code can be altered correspondingly.

Currently it sufficed to record only the current and last position and the distance between the various parts of the skeleton, specifically hands, elbows, and body. For something that requires more careful movement tracking (such as software used in surgery or engineering), it would be wise to save the positions of the user and the position of his/her limbs more often, recommended would be somewhere between 30-50 times, so that there would be no chance of an accidental gesture happening. A slide changing program is in its essence quite simple and not mission-critical. The worse thing that could happen is that the user must perform the swipe gesture one more time. That is also why the swipe gesture is quite general, different people may perform the swipe gesture a little different.

It is worth mentioning that it is also possible to track joint angles. This possibility is not used in this work, but again, for some applications that require precision, this could definitely prove useful.

4.2. Workflow

The application can in its current realization handle one user standing in front of the camera. Before the user can start using the gesture recognition capabilities of the application, he or she needs to initialize Kinect for it to track the person and collect necessary data.

The program is visualized *via* a graphical user interface with some buttons and two image streams created from Kinect sensor color and depth data. The color image is an undmodified image that shows one-to-one what the Kinect camera sees. Additionaly it displays some hand position info at the bottom of the color image. The depth image allows to see that when the user, or one of the limbs is too close to the sensor, the area turns black indicating that no data could be obtained.

The picture below shows the GUI. The leftmost area contains all the buttons and GUI-specific objects. To the right of the buttons are two image streams. The one on the left is the color image and the one on the right shows depth data.

Figure 5. Graphical User Interface.

The buttons and their meanings:

- Connect to Kinect – initializes Kinect and starts collecting data.
- Adjust Kinect – enables to change the tilt of Kinect.
- Current Angle – displays the current angle at which the Kinect is placed.
- Select presentation file format – a separator under which there are possible radio button options for file formats, which are necessary to send correct input signals to the operating system.
- Choose dominant hand – groups radio buttons for choosing the dominant hand.
- Status info - text area for status info.

To use the program with Kinect, You must first click „Connect to Kinect“ button. This will try and initialize Kinect. If successful, then two image streams will be displayed and some general info about initialization status will be written into the status info text area. If unsuccessful, the reason for failure will be shown in the status info area.
There are options for selecting between Powerpoint, PDF, or Word file depending on what file You want to present. Then the work with gesture recognition can start. For registering a valid

movement, make the presentation file window active and face the camera. Then perform one of the gestures described above.

Thanks to Kinect SDK, the tilt motor is also utilized. It is possible to change angles from -27 to +27 [5]. With the help of the live image stream, this makes it possible to optimize the lens angel so that the head of the user would be seen without manually handling Kinect (manually tilting the Kinect can damage the sensor). The Kinect tilt motor is not recommended to be moved more than once *per* second and let it rest at least 20 seconds after 15 consecutive changes [28]. This logic is written into the program, and when the tilt functionality is utilized 15 times, then the next time (unless the user waits for 20 seconds) someone tries to change the angle using the GUI, status text area will contain a warning message about not abusing the tilt motor.

5. Similar existing solutions

There is various software available for Kinect and Windows with the main goal to help make presentations through a natural user interface. A few of them are proprietary with a license available for purchase. Others are free to download and experiment with. The author managed also to find two thesis works on the same subject (a bachelors level, and the second one a masters level work).

5.1. So Touch Air

So Touch Air [29] is a product for Kinect and Windows 7 for interacting with a computer through a natural user interface implemented with Kinect SDK. It is available as a free trial from *software.informer*, and also for buying from a company called Nyu Systems. The project seems to have been closed down, because the download link in the official page does not lead to anything. As far as I could tell from the videos uploaded by the developing part, the program is a specially designed interface application to control presentation. It is not a solution that works side by side with traditional presentation software like PDFs or powerpoints, but is in itself a complete presentation solution. It extracts pages/slides from PDFs, powerpoints, images, and integrates them into the natural user interface.

5.2. PowerPoint Presenter for Kinect for Windows

An application developed by Evoluce. PowerPoint Presenter for Kinect for Windows [30] is a proprietary softare product that comes closest from the licensed solutions found to this thesis work. Like the So Touch Air, and unlike the program created with this thesis, it is a full-fledged interface application. It loads slides from files to the interface for presentint and allows to manipulate them through the application itself. This is different from the work done here, where the application and presentation are separate processes. It provides more features than perhaps necessary in a simple educational presenter, where the main objective is to change slides quickly while giving a lecture to a group of people in a hall.
The license is available for purchase for euro 29,90.

5.3. Ubi Interactive

Ubi Interactive [31] has created a solution that provides a touchscreen meets gesture recognition approach for providing an augmented reality experience. From what the author learned from browsing their homepage, it is possible to use a pen, finger or gestures to interact with a computer running Windows 8. This is not only for making presentation, but using all kinds of different Windows 8 applications. The price is from 149 dollars a license to 1499 a license.
For a simple natural user interface option for making presentations in an academic enviroment, this seems like a bit of an overkill compared to the application created by the author in the thesis at hand.

5.4. Non-commercial solutions

Unlike official products that cost money and are provided by professional software companies, there also exist solutions made in universities, or by independent developers for learning reasons, or just for fun. These are usually quite basic experiments that just try to show that it is indeed possible to communicate with Your computer with the help of Kinect motion sensing.

I was able to find two thesis works, one Bachelor thesis made in Chicage State University (CSU) [32] and the other a Master thesis made in Massachusetts Institute of Technology (MIT) [32]. The first one discusses the possibilites of using gestures to control a Windows computer and also suggests making presentations as one of the possibilities. A working solution for controlling a presentation does not seem to have been implemented, but the idea is presented.

The second thesis work is a master's thesis that describes the design and implementation of a speech and gesture recognition system used to control a PowerPoint presentation using the Microsoft Kinect. It is written in C# and takes a more sophisticated approach to tracking movement and identifying gestures. It's main objective does not seem to be to develop an application, but analyze the details of gesture recognition by using PowerPoint presentations as an example.

The first place where a lot of developers interest in using Kinect SDK (or any other software development kit) turn to, is the web and various forum-like sites. Some example solutions that were found browsing with a similar functionality to this thesis work include:

- An example solution found in CodePlex. It is similar to the one created with this thesis. It provides simple gesture recognition capabilities using the same principles as this work. It is written in C#. [33]
- Microsoft has itself been a supporter of the idea of having Kinect helping out when making presentations. For example, Microsoft Worldwide Partner Conference 2012 included some examples. [34]
- KineSis is a bigger project designed to help in presenting documents with Microsoft Kinect. The users have the possibility of opening documents (Microsoft Office documents, images and plain text), and based on gestures, to control the presentation (move to next slide/page, scroll, zoom). Another feature is the intervention in current slide/page by focusing or highlighting the important elements from presentation (painting). It is similar to Ubi Interactive solution in that it isn't just a simple way to make presentations, but digs in to the document presenting more deeply, for example it allows to zoom in to little areas and highlight them. [35]

6. Where to go from here

There is definitely potential for further development. Especially as a free-source implementation. In this section I try and analyze what could be done if work would continue with this application.

Firstly, it must be reminded that all the possible features, functions and ideas described in this section were not implemented in this work. Some of these possibilites were explored a bit, but nothing substantial was created. The current program is pretty basic, but with a little more effort it could easily become a fully functional free-source (because all the necessary librarys, including Kinect SDK, is free for everyone) easy-to-use presentation program.

6.1. Advanced tracking

Currently the program can handle a single person standing in front of the camera. Kinect SDK provides options for dealing with more people, but due to the fact that the work was done with limited access to human resource, this functionality has not been implemented. It would have required testing with a partner that would have been freely available whenever necessary. The SDK provided tracking ID feature [9] was tested, so if necessary, it would not be very hard to make the program functional with more than one people standing in the presenter area. This would allow a duo of presenters to switch the role of presenter at random.

Another option that would make tracking a bit more advanced, would be to automate the moving of the tilt motor. Perhaps if it is recognized that the head of the presenter is a bit out of the frame, the program would automatically elevate the motor. The specification warns about moving the camera too often. The tilt motor could get damaged. That would require careful handling. Especially in cases where the presenter moves a lot.

6.2. Adding audio capabilities

Kinect also has an audio stream [37] that maybe could have been implemented by opening or interacting with the program through voice control, but due to the fact that the area of use is in a lecture hall with potentially hundreds of people, the noise and identifying the correct person talking would have required a lot of testing. To locate the right person is indeed possible through beamforwarding, source localization, and echo cancellation, but these features are quite complex and thus were not explored personally in the context of this thesis work.

6.3. Making it OS independent

Currently the program ahieves the slide changing and other features through interaction with the Microsoft Windows operating system and keyboard events. This is not portable to other operating systems.

Visual Studio provides project templates that can be used to create application-level add-ins for Microsoft Office PowerPoint [38]. The alternative to Microsoft Office, OpenOffice, also has an SDK [39]. Foxit Reader and Adobe PDF Reader are two of the most famous PDF softwares. Both have proprietary SDKs [40, 41] for developrs with a limited time free trial option. All of these could be used to implement a platform independent solution.

6.4. Adding supprt for sitting down mode and touchscreen-like interaction

Currently the program can handle presentations being made standing up and facing the Kinect. The SDK additionally allows to use sitting mode. There is even an open framework that defines a common protocol and API for tangible multitouch surfaces, it is called TUIO [42]. Sitting behind the computer may hide the person from the Kinect camera. This would mean that more consideration should be put into the positioning of the device. Using only one Kinect, this would limit the presenting area, but with perhaps two Kinect devices, one mounted somewhere above the computer, and the other on one side of the person, support for sitting down, and tourchscreen-like interaction would be possible to implement.

6.5. Enhancing user comfort

Natural user interface devices are known to go with software with self-learning capabilities built in. The user would learn to use the program not by studying defined rules, but through active interaction. This could be enabled by enhancing the code to use some kind of video recording. This would make it more comfortable for users. Presenters would just run the program and the software could adapt to the specific user and their needs automatically.

7. Conclusion

The goal of this thesis was to find out if it is possible to create a useful Kinect-based application that can use gesture recognition for providing a practical way to make presentations through a natural user interface.

The first thing was to see if it was at all possible for a single person to make Kinect react to movements and gestures and use gathered data in a practical manner without overwhelming and unreasonably difficult work. The thesis at hand answered this question positively, all of what Kinect and its associated SDK offers is easy enough to learn and can be effectively used for making software by individuals interested in natural user interface applications.

The next step was to achieve a working solution that would actually be considered as a viable option when making presentations. With the help of Visual Studio and various APIs, the end result is considered to be successful. The application created achieves this goal. The software can assist in making presentations without being overly complex.

The conclusion is that the biggest impediment in making practical NUI-applications that can help in practical work is acquiring a Kinect device. All of the support and resource out there is available and it is only up to developers to make use of them for providing sought-after applications.

List of resources

[1] R. Wang, „Augmented Reality with Kinect“, Packt Publishing, Birmingham, 2013.

[2] Kinect SDK. [Online]. [Accessed 30.04.2015]. Available: https://www.microsoft.com/en-us/kinectforwindows/

[3] Visual Studio. [Online]. [Accessed 12.05.2015]. Available: https://www.visualstudio.com/en-us/features/ide-vs.aspx

[4] Wikimedia Foundation Inc, „Kinect“. [Online]. [Accessed 15.05.2015]. Available: http://en.wikipedia.org/wiki/Kinect

[5] Microsoft Developer Network, „Kinect for Windows Sensor Components and Specifications“. [Online]. [Accessed 15.05.2015]. Available: https://msdn.microsoft.com/en-us/library/jj131033.aspx

[6] OpenKinect. [Online]. [Accessed 27.05.2015]. Available: http://openkinect.org/wiki/Main_Page

[7] Microsoft Developer Network, „System Requirements“. [Online]. [Accessed 16.05.2015]. Available: https://msdn.microsoft.com/en-us/library/hh855359.aspx

[8] Microsoft Developer Network, „Kinect for Windows Programming Guide“. [Online]. [Accessed 09.05.2015]. Available: https://msdn.microsoft.com/en-us/library/hh855348.aspx

[9] Microsoft Developer Network, „Skeletal Tracking“. [Online]. [Accessed 13.05.2015]. Available: https://msdn.microsoft.com/en-us/library/hh973074.aspx?f=255&MSPPError=-2147217396

[10] Microsoft Developer Network, „Tracking Users with Kinect Skeletal Tracking“. [Online]. [Accessed 11.05.2015]. Available: https://msdn.microsoft.com/en-us/library/jj131025.aspx

[11] Microsoft Developer Network, „Coordinate Spaces“. [Online]. [Accessed 14.05.2015]. Available: https://msdn.microsoft.com/en-us/library/hh973078.aspx

[12] Wikimedia Foundation Inc, „Microsoft Visual Studio“. [Online]. [Accessed 24.05.2015]. Available: http://en.wikipedia.org/wiki/Microsoft_Visual_Studio

[13] FreeGLUT [Online] [Accessed 05.05.2015]. Available: http://freeglut.sourceforge.net/

[14] R. Wang, GLUtilities source code. [Online]. [Accessed 05.05.2015]. Available: https://github.com/xarray/augmented-reality-with-microsoft-kinect/tree/master/common

[15] Microsoft Developer Network homepage. [Online]. [Accessed 12.05.2015]. Available: https://msdn.microsoft.com/en-us/default.aspx

[16] Microsoft Developer Network, „Setting Up a Kinect Sensor“. [Online]. [Accessed 30.04.2015]. Available: https://msdn.microsoft.com/en-us/library/hh855356.aspx

[17] Microsoft Developer Network, „Face Tracking“. [Online]. [Accessed 11.05.2015]. Available: https://msdn.microsoft.com/en-us/library/jj130970.aspx

[18] Microsoft Developer Network, „Skeleton Tracking With Multiple Kinect Sensors“. [Online]. [Accessed 25.05.2015]. Available: https://msdn.microsoft.com/en-us/library/dn188677.aspx

[19] GLUT documentation. [Online]. [Accessed 22.05.2015]. Available: https://www.opengl.org/documentation/specs/glut/

[20] wxWidgets. [Online]. [Accessed 14.05.2015]. Available: https://www.wxwidgets.org/

[21] OGRE. [Online]. [Accessed 15.05.2015]. Available: http://www.ogre3d.org/

[22] GLUI homepage. [Online] [Accessed 22.05.2015]. Available: http://glui.sourceforge.net/

[23] Paul Rademacher „GLUI. A GLUT-based User Interface Library manual“. [Online]. [Accessed 22.05.2015]. Available: https://www.cs.unc.edu/~rademach/glui/src/release/glui_manual_v2_beta.pdf

[24] Qt. [Online]. [Accessed 22.05.2015]. Available: http://www.qt.io/

[25] Wikimedia Foundation Inc, „Microsoft Developer Network“. [Online]. [Accessed 24.05.2015]. Available: http://en.wikipedia.org/wiki/Microsoft_Developer_Network

[26] Microsoft Developer Network, „Keyboard Input“. [Online]. [Accessed 08.05.2015]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ms645530(v=vs.85).aspx

[27] Microsoft Developer Network, „Functions“ [Online]. [Accessed 20.05.2015]. Available: https://msdn.microsoft.com/en-us/library/hh855370.aspx

[28] Microsoft Developer Network, „NuiCameraElevationSetAngle“ [Online]. [Accessed 22.05.2015]. Available: https://msdn.microsoft.com/en-us/library/nuiimagecamera.nuicameraelevationsetangle.aspx

[29] So Touch Air. [Online]. [Accessed 20.05.2015]. Available: http://www.nyu-systems.com/your-multi-touch-applications/so-touch-air-presenter-plus%EF%BB%BF-2/

[30] Evoluce, „PowerPoint Presenter For Kinect For Windows“. [Online]. [Accessed 19.05.2015]. Available: http://www.evoluce.com/kinect-powerpoint-presenter.htm

[31] Ubi Interactive. [Online]. [Accessed 19.05.2015]. Available: http://www.ubi-interactive.com/

[32] T. Osunkoya, J. C. Chern, „Gesture-Based Human-Computer-Interaction Using Kinect for Windows Mouse Control and PowerPoint Presentation“, Bachelor thesis, CSU, Chicago, 2013. [Accessed 23.05.2015]. Available: micsymposium.org/mics_2013_Proceedings/submissions/mics20130_submission_18.pdf

[33] Chang, S. M., „Using Gesture Recognition to Control PowerPoint Using the Microsoft Kinect“, Master thesis, MIT, Massachusetts, 2013. [Accessed 24.05.2015]. Available: http://dspace.mit.edu/handle/1721.1/85410

[34] J. Blake, „Kinect PowerPoint Control“ source code. [Online]. [Accessed 22.05.2015]. Available: https://kinectpowerpoint.codeplex.com/

[35] D. R. Sy, „How to Use Kinect with PowerPoint“. [Online]. [Accessed 23.05.2015]. Available: http://meetdux.com/2012/06/21/kinect-powerpoint/

[36] KineSis. [Online]. [Accessed 22.05.2015]. Available: http://code.google.com/p/kinesis/

[37] Microsoft Developer Network, „Audio Stream“. [Online]. [Accessed 22.05.2015]. Available: https://msdn.microsoft.com/en-us/library/jj131026.aspx

[38] Office automation. [Online]. [Accessed 17.05.2015]. Available: https://msdn.microsoft.com/en-us/library/bb772069.aspx

[39] OpenOffice SDK. [Online]. [Accessed 17.05.2015]. Available: http://www.openoffice.org/download/sdk/

[40] Foxit SDK. [Online]. [Accessed 17.05.2015]. Available: http://www.foxitsoftware.com/products/sdk/PDFsdk/

[41] Adobe SDK. [Online]. [Accessed 17.05.2015]. Available: http://www.adobe.com/devnet/pdf/library.html

[42] TUIO. [Online]. [Accessed 18.05.2015]. Available: http://www.tuio.org/