Tartu University

Faculty of Science and Technology

Sandra Demitševa

GESTURE BASED COMPUTER CONTROLLING USING KINECT CAMERA

Bachelor's thesis

Supervisor: Assoc. Prof. Gholamreza Anbarjafari

Tartu

2015

Abstract

The purpose of this thesis is to research the possibilities of human-computer interaction and create an application it is hoped to achieve a more natural user interface when controlling a Windows PC. The solution uses a set of technologies provided by Microsoft and relies on a 3D motion sensor that can track humans and objects in its field of view. The thesis consists of three parts. The first part gives an overview of the sensor that is used. The second part describes the software development kit. The third part specifies the work that has been done and the finger tracking library used.

Contents

List of Figures

List of Tables

Abbreviations

HMI – human-machine interaction

3D – three dimensional

NUI – natural user interface

PC – personal computer

USB – universal serial bus

SDK – software development kit

CMOS – complementary metal-oxide-semiconductor

HD – high definition

C# – C sharp

DLL – dynamic link library

dB – decibel

OS – operating system

API – application programming interface

Introduction

The goal of developing HMI (human-machine interaction) devices is creating interfaces that are efficient and user friendly. After starting the wide use of touchscreens in information appliances, rather than using intermediate devices like a mouse or touchpad, interaction with what is displayed, has become more natural. Gesture recognition is a way to interpret human gestures through mathematical algorithms, therefore computers can be able to understand human body language. Using vision based hand gesture recognition makes human-computer communication even more intuitive and unobtrusive.

In the progress of computing and display technologies settling with existing ways of technology may become an obstacle for further development. Gesture recognition could mean that conventional user interfaces like a mouse will become obsolete. Moving the cursor will be done by just pointing at something on the screen without the need to wear, hold or touch anything. Using vision based hand recognition provides effortless interaction with 3D (three dimensional) objects, unlike using a mouse, because it can properly emulate the three dimensions of space.

There is a vast range of devices where gesture recognition can be incorporated into – from personal computers, televisions to tablets and smartphones. Vision based gesture recognition has many applications. For example it makes 3D design and visualization faster and more intuitive. Doctors can scroll through medical files sterilely during an operation. It can also be helpful for people with physical disabilities or who have carpal tunnel injuries. Not to mention gaming will change completely.

Due to the complexity of the motion and structure of the human hand and difficulty of locating fingertips, vision based recognition and tracking is extremely challenging. Earlier approaches used position markers and colored gloves to make recognition easier, but it could not be considered as a natural interface. The new technique in human-machine interaction uses nothing but bare hands as the input media.

Problem statement

The goal is to create an application that can establish a touchless control of a Windows PC. This removes the need to use a mouse or keyboard to navigate the operating system and makes the user experience more natural. For this to be possible it is essential for the computer to be able to “see” the user, track their hand and recognize hand postures to simulate mouse events. All of this has to happen in real-time, without delay. The sensor that is going to be used can track up to 25 body joints, but cannot recognize fingers.

Kinect sensor

The Kinect sensor, originally known as Project Natal, is a motion-sensing device developed by Microsoft. The idea behind it was replacing the conventional gaming controller by removing it entirely. The sensor is a horizontal device with a color camera, depth sensors and a set of microphones (Figure 1) designed to be positioned lengthwise above or below the video display. The Kinect provides a NUI (natural user interface) that can detect body motion, gestures, spoken commands and even facial recognition. This enables users to interact with their console or PC (personal computer) without using hand-controlled devices. [1]

A particularly accurate 3D view of the surroundings was achieved with the first sensor by projecting infrared light beams into a space and measuring the changes in the patterns. The biggest challenges were getting the device to track the bodies of the users as they move and recognizing the same person regardless of appearance changes and rapid movements. All of this had to happen without delay in real-time. [2] [3]

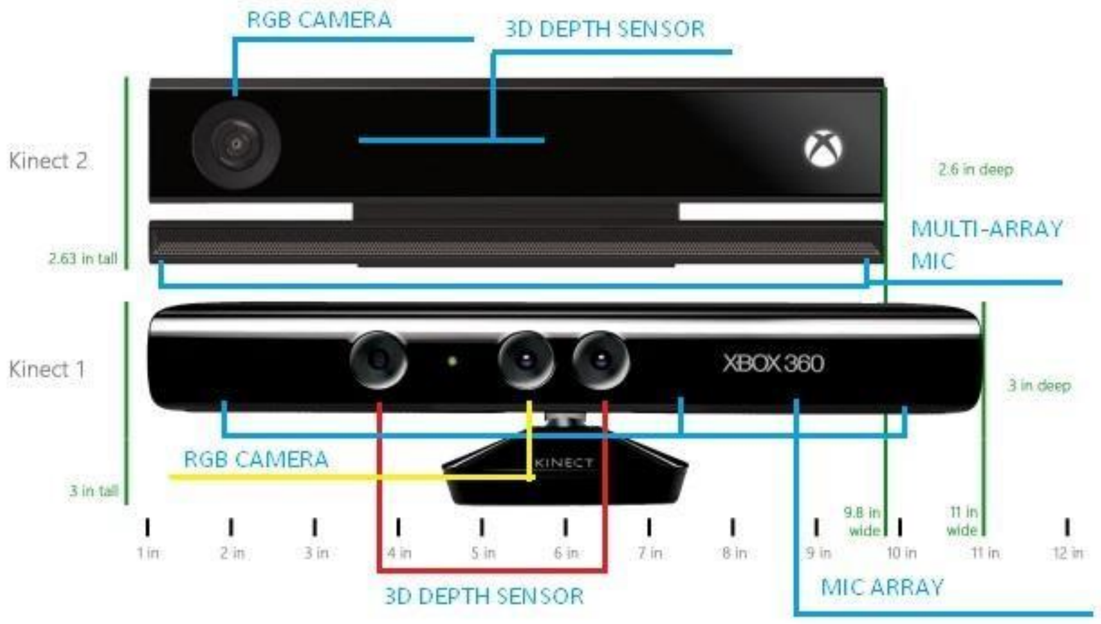


Figure 1: Kinect for Xbox 360 and Kinect for Xbox One

Kinect for Xbox 360

Developed for the Xbox 360 console and released it is the first in the line of Kinect sensors. Holds the Guinness World Record for being the fastest-selling gaming peripheral. In its first 60 days on sale from November 4th, 2010 to January 3rd, 2011 an average of 133,333 units per day were sold. [4]

Kinect for Xbox 360 was built to track players that are up to 4,5 meters away from the sensor, but fails to track objects that are closer than 0,8 meters. [5]

Kinect for Windows v1

Kinect for Windows was released on February 1st, 2011and is primarily a developing device based on the same core technology as Kinect for Xbox 360. The Kinect for Windows sensor comes with an external power supply for additional power and a USB (universal serial bus) adapter to connect with a PC or tablet. The Kinect for Windows sensor does not work with the Xbox gaming console. For Kinect for Windows v1 new firmware was added which enabled Near Mode tracking. This gave support to track objects as close as 40 centimeters in front of the device without losing accuracy or precision. [6]

On June 16th, 2011 Microsoft announced the release of its SDK (software development kit) for non-commercial use. The Kinect for Windows was designed to expand the purposes of the Kinect sensor so that it could be used to develop software for real-life applications with human gestures and voice commands. Since February 1st, 2011, after the release of the commercial version of Kinect for Windows SDK, there is also the possibility to create apps for commercial distribution through the Windows Store, by using C++, C#, Visual Basic or any other .NET language of Windows Store projection. [1]

Kinect for Xbox One

Kinect for Xbox One was released November 22nd, 2013 and had major upgrades compared to its predecessor for Xbox 360 (Table 1). The new Kinect also uses infrared to read its environment, but uses a wide-angle time-of-flight camera and processes 2 gigabits of data per second. The sensor can track without visible light by using an active infrared sensor. Kinect for Xbox One has no separate Near Mode, but has greater accuracy (resolution is 20 times greater) and an improved field of view, reducing the amount of distance needed between the player and the sensor for optimal Kinect performance. The new Kinect no longer contains a tilt motor and the physical design of the new sensor is more similar to the architecture of the Xbox console. [9] [10]

Table 1: Kinect for Windows v1 comparison to Kinect for Windows v2. [2] [6] [7] [8]

Feature	**Kinect for Windows v1**	**Kinect for Windows v2**
Color camera	640x480 pixels @ 30Hz	1920x1080 @ 30Hz
Depth camera	320x240 pixels @ 30Hz	512x424 pixels @ 30Hz
Max depth distance	4.5 m	4.5 m
Horizontal field of view	57 degrees	70 degrees
Vertical field of view	43 degrees	60 degrees
Tilt motor	Yes	No
Skeleton joints defined	20	26
Full skeletons tracked	2	6
USB standard	2.0	3.0
Supported operating system	Windows 7, Windows 8	Windows 8

Kinect for Windows v2

The second-generation Kinect for Windows was released on July 15th, 2014 alongside the Kinect for Windows SDK 2.0. It is intended for those developing Kinect-enabled software for Windows 8 and Windows 8.1. Higher depth fidelity makes it easier to see objects more clearly and in 3D (Figure 2). [9]

October 22nd, 2014 Microsoft introduced an adapter kit that enables users to attach their Kinect for Xbox One sensors to Windows PCs and tablets over USB 3.0. With the adapter Kinect for Windows v2 and for Xbox One perform identically. [11]

April 2nd, 2015 Microsoft announced that they will no longer produce separate Kinect for Windows sensors and will focus on Kinect for Xbox One sensors and Kinect Adapter for Windows units. [12]

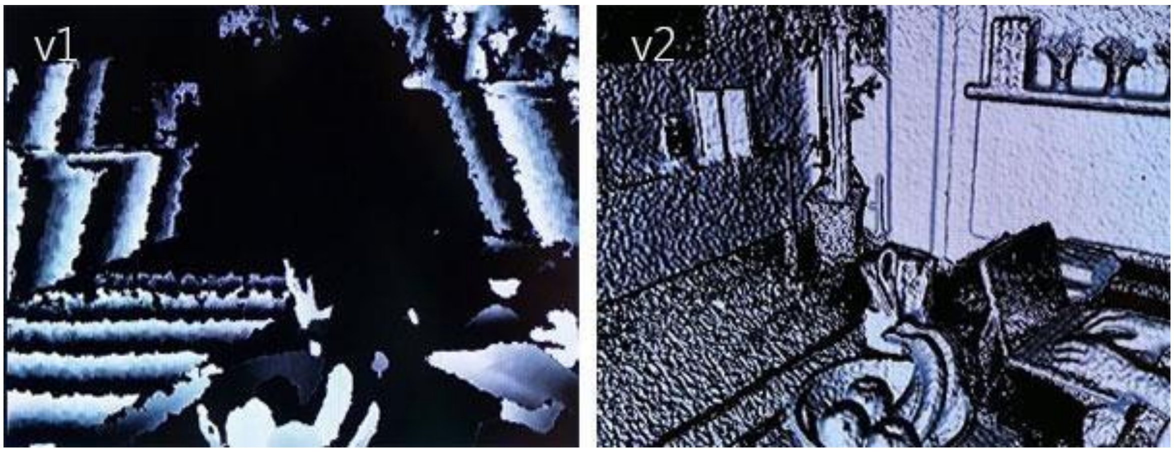


Figure 2: Depth data

Kinect architecture

Depth data

The Kinect uses the depth information to track the position of people in front of the sensor. The depth sensor consists of an infrared laser projector and monochrome (black and white) CMOS (complementary metal-oxide-semiconductor) sensor. The CMOS sensor captures light and converts it into electrical signals. [13]

Inferring body position for Kinect for Xbox 360 works by first computing a depth map using structured light and then using machine learning. The depth map is constructed by analyzing a speckle pattern of infrared laser light that is invisible to the eye. Microsoft licensed this technology from a company called PrimeSense. The depth computation is done by the PrimeSense hardware built into Kinect. It works by projecting a known pattern onto the scene (Figure 4) and inferring depth from the deformation of that pattern (structured light) through comparing the image from the camera with the pattern it knows it is displaying. The dots are arranged in a pseudo-random pattern that is hardwired into the sensor. The infrared sensor is fitted with a filter that keeps out ordinary light, which is how it can see just the infrared dots, even in a brightly lit room.

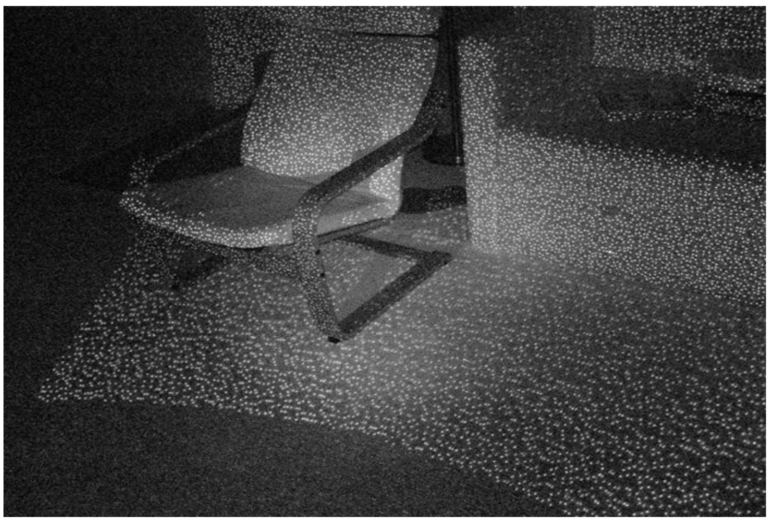


Figure 3: Set of infrared dots scattered on the scene

The first Kinect combines structured light with two classic computer vision techniques: depth from focus and depth from stereo. Depth from focus uses the principle that objects that are more blurry are further away. The Kinect uses a special lens with different focal length in x- and y- directions. A projected circle then becomes an ellipse whose orientation depends on depth. Depth from stereo – if you look at the scene from another angle, objects that are close get shifted to the side more than objects that are far away. The Kinect analyzes the shift of the speckle pattern by projecting from one location and observing from another. [14]

As the reflecting surface gets farther away from the camera, the dots do not look smaller and closer together because the camera and the laser projector are *epipolar rectified*. This means that the camera and projector have matching fields of view. As the reflecting surface gets farther away from the sensor, the light from the laser is a cone of light that is getting larger as you get farther from its source. [15]

Body parts are inferred by machine learning – feeding the computer data in the form of millions of images of people and it learns how to understand them. Microsoft gathered pictures of people in many different poses and ran the data through huge clusters of computers (Figure 5). The recordings do not tell the computer anything useful about joints and limbs, so programmers hand-coded the raw data to label each body part. These marked-up images make tens of terabytes of information that the Microsoft's computer farms sift through coming up with probabilities and statistics about the human form.

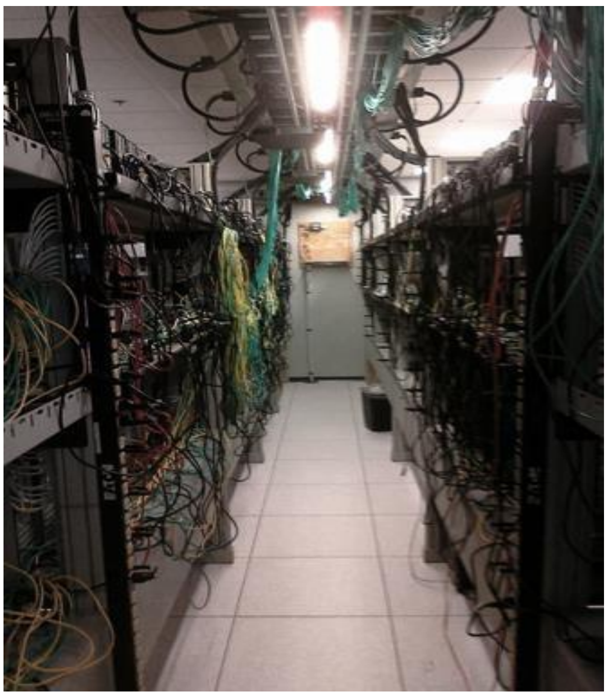


Figure 4: Microsoft's clusters of computers are the "learning brain" that feeds all Kinects

The Kinect guesses which parts of the body are which based on all of its experience with body poses. Then, based on the probabilities assigned to different areas, Natal comes up with possible skeletons that could fit the body parts. Ultimately it selects the most probable one and outputs the data into a simplified 3D avatar shape. To get proper coordinates, the sensor calculates the three views of the same image: front, left and top view, by which it defines the 3D body proposal. Analyzing the data is done 30 times in a second. [16]

The new Kinect has a time-of-flight camera. It consists of the following:

- Illumination unit: Illuminates the scene using infrared.
- Optics: A lens gathers the reflected light and images the environment onto the image sensor (focal plane array). An optical band-pass filter only passes the light with the same wavelength as the illumination unit. This helps suppress non-pertinent light and reduce noise.
- Image sensor: Each pixel measures the time the light has taken to travel from the illumination unit to the object and back to focal plane array.
- Driver electronics: Controls and synchronizes the illumination unit and image sensor.
- Computation/Interface: Distance is calculated directly in the camera and no manual depth computation is required. [17]

The basic idea is that the round-trip time is measured in which the photons (quantum in which light and other forms of electromagnetic radiation are transmitted) are emitted by the sensor and reflected back and this time is used to calculate the distance. Because the sensor does this measurement with every one of its quarter million pixels at the same time, it can generate a 3D image with an amazing level of accuracy. This technology also improves the ability to see objects more clearly, recognize smaller objects and enhances the stability of body tracking, even fingers on the hands. [18] [19]

The sensor can see from 0.5 meters to 4.5 meters by default with the optimal distance for body detection is from 0.8 to 3.5 meters. The depth sensor can also see from 4.5 to 8 meters, but body detection does not work in this extended range. The field of view of the new sensor is 60% wider. As a result users can be closer to the camera and still in view and the camera is effective over a larger total area. Also the remarkable improvement in resolution has helped make considerable advancements in facial recognition with the Kinect sensor. [9]

The chip and circuitry design of the time-of-flight are very challenging making the sensor costly. 3DV Systems found a way to lower the costs dramatically and planned to release a RGB-Depth sensor, called ZCam for under 100 dollars, but the company was bought by Microsoft. [20]

Color camera

The color camera is responsible for capturing and streaming the color video data. Its function is to detect the red, green and blue colors from the source. It simply acts like a basic webcam and records the room. The first Kinect captures images 30 frames per second and projects at 640x480 resolution.

The color camera of the Kinect for Xbox One captures 1080p HD (high definition) video at 30 frames per second. The value of frames per second may vary depending on the resolution used for the image frame. [21]

New active infrared

The new Kinect has an active infrared, which gives it the ability to see in the dark. The variations in the room lighting can make it difficult to see who is in the frame, so the active infrared removes ambient room lighting.

Using the feed from the high definition color camera and active infrared, it is possible to detect a heart rate by looking for subtle variations in color and intensity in the skin of the face. This can be used in exercise scenarios and when measuring exertion. [22]

Microphone

The Kinect for Xbox 360 has four microphones in its sensor bar and they are pointed down to pick up sound waves more effectively. Kinect uses these microphones to help determine from where in a room a particular voice is coming from. This works because sound takes much more time to travel through air then light. When you speak to the Kinect your voice will arrive at each microphone at different times, because they are a slightly different distance away from the sound source. In addition to determining the direction of sound signals, it isolates them from background noise.

The microphone array can cancel 20dB of ambient noise, which improves audio fidelity. Voice commands require calibration so that the Kinect knows the general audio levels of the surroundings. If you rearrange your furniture or add a carpet, then you will have to recalibrate, because sound bounces around the room differently. [23] [24]

The new Kinect microphone array enables more extensive voice command support then the old version. It is possible to programmatically direct the microphone array. When the Xbox One sensor is in sleep mode, the microphone can still stay active, so that the console can wake up by using a voice command. The new Kinect sensor detects audio input from +50 and -50 degrees in front of the sensor. By default the sensor tracks the loudest audio input. [8]

Tilt motor

The Kinect for Xbox 360 has a tilt motor which can be used to change the angle of the camera without physical interaction. The sensor for Xbox One does not have this feature.

Sensor positioning

- The sensor works best when it is positioned between 0.6 and 1.8 meters off the floor. In smaller rooms the sensor should be as close to 1.8 meters as possible.
- The Kinect should stand near the edge of a flat, stable surface to see the feet and make sure no nearby objects are obstructing the field of view.
- The sensor should not be positioned near a speaker, vibrating surface or glass doors.
- Avoid positioning the sensor in direct sunlight. The depth camera works in all lighting situations, but with large amounts of natural light skeleton tracking becomes less reliable. Ideally the light source should be from behind the sensor.

The placement requirements of both Kinects are similar. With the Kinect for Xbox One, users are able to stand closer to the sensor and it offers a wider field of view. For the old Kinect it was important to place the sensor so it can tilt and automatically adjust freely. [25] [26] [27] [28]

Hardware and software requirements

A short description of software and hardware requirements that are needed to write software for the Kinect sensor.

Kinect for Xbox 360 or Kinect for Windows v1

Minimal hardware requirements for the PC or tablet used with the Kinect sensor:

- 32-bit (x86) or 64-bit (x64) processor
- Dual core 2.66-GHz or faster processor
- Dedicated USB 2.0 bus
- 2 GB RAM
- DirectX 9 capable video card running at 1024x768 or higher resolution

The Kinect for Windows sensor comes with an external power supply and a USB adapter to connect with the system. Using the sensor from the Xbox 360 console it will also require a separate power supply so it can power up the motor, camera, infrared sensor etc. It can be bought independently.

Minimal software required to be installed to access the capabilities of the sensor:

- Supported operating systems are Windows 7, Windows Embedded Standard 7, Windows 8 and Windows 8.1
- Kinect for Windows SDK v1.0 through v1.8
- Microsoft Visual Studio 2010 or Visual Studio 2012 (any edition)
- .NET Framework 4 (installed with Visual Studio 2010), .NET Framework 4.5 (installed with Visual Studio 2012) [29]

The Kinect for Windows sensor will also work on Windows operating systems running in a virtual machine such as Microsoft HyperV, VMWare and Parallels.

Kinect for Xbox One or Kinect for Windows v2

Minimal hardware specifications:

- 64-bit (x64) processor
- Physical dual-core 3.1 GHz or faster processor
- Dedicated USB 3.0 bus
- Graphics card, that supports DirectX 11

Minimal software requirements:

- Windows 8, Windows 8.1 or Windows Embedded 8
- Kinect for Windows SDK 2.0
- Microsoft Visual Studio 2012 or Visual Studio 2013
- .NET Framework 4.5 [30]

Kinect for Windows SDK

The Kinect for Windows SDK is a free download. It is used to write programs that use Kinect devices. The SDK contains all the drivers needed to link a Kinect to your computer and provides a set of libraries that can be added to the software so it can interact with the sensor (Figure 5). It gives direct access to low-level video and depth signals and the powerful library features built into the SDK make it possible for a program to identify and track users. Building applications can be done using C# or Visual Basic (managed code) or C++ (unmanaged code). [24]

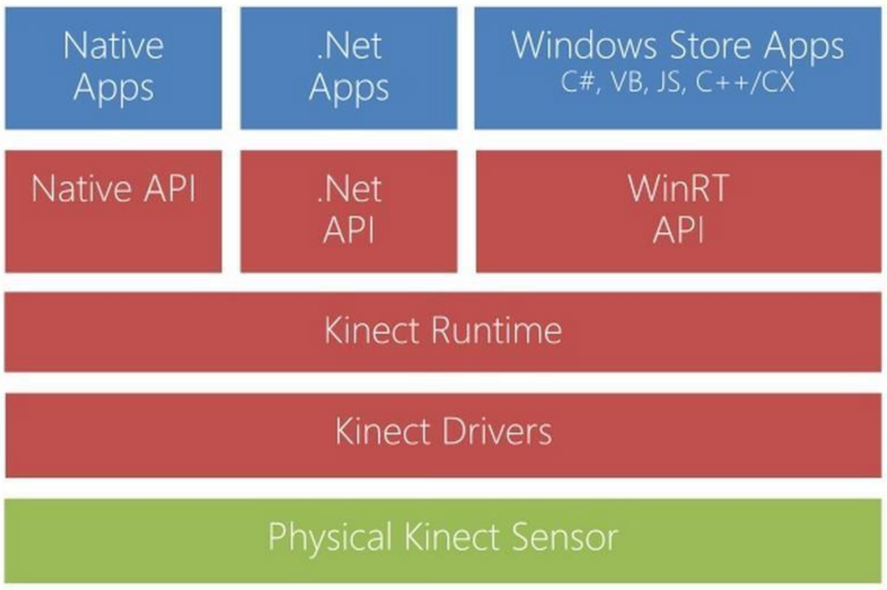


Figure 5: SDK architecture. Interaction between application and different layers of components

The first two versions of Kinect for Windows SDK (Beta 1 and Beta 2) were non-commercial releases and were meant for hobbyists. On February 2012 the first commercial version (v1.0) was released. [4] The SDK also includes samples of good practices for using a Kinect sensor and example codes that breaks down samples into user tasks. [31]

The Kinect SDK uses depth information to track the position of people in the field of view of the sensor. Skeletons can be tracked whether the user is standing (25 joints) or seated (10 joints). With the SDK v2.0 the tracked body positions are more anatomically correct and stable (Figure 6). The Kinect skeleton is made up of 25 joints – including new joints for hand tips, thumbs and shoulder center. Each joint is positioned in 3D space relative to the Kinect sensor. The position of each joint is given as an offset from the Kinect sensor. From the point of view of the user the x axis extends to the right, the y axis upward and z axis is oriented from the sensor to the user and expressed in meters. [32]

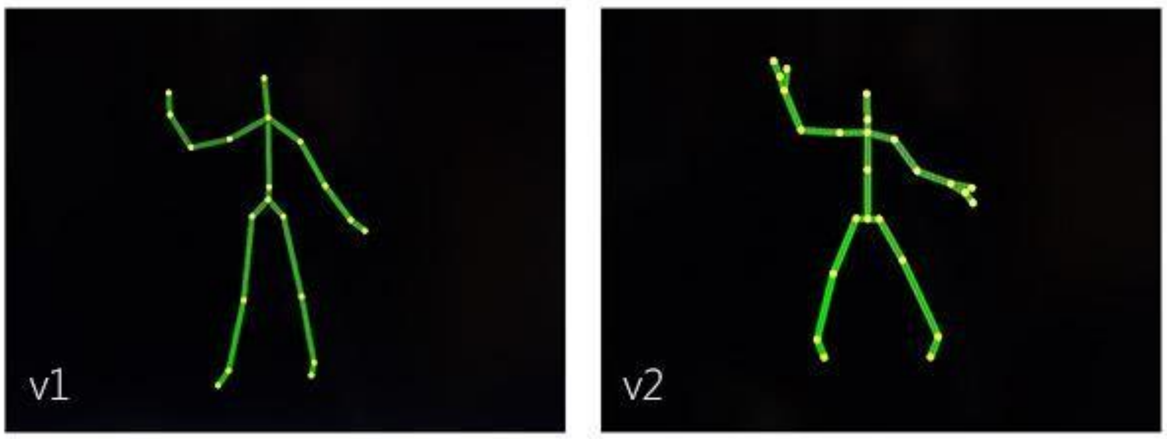


Figure 6: Comparison of body tracking precision for Kinect for Windows SDK v1.0 and v2.0

With the new SDK:

- It is possible to create Windows Store applications and offer them to consumers.
- It also offers support for Unity Pro users for building and publishing apps.
- Multiple apps can now access a single sensor.
- Representation of a person’s face is more lifelike.
- Visual Gesture Builder enables developers build their own gestures that can be recognized decreasing the time to prototype and test solutions. [33]

Work description

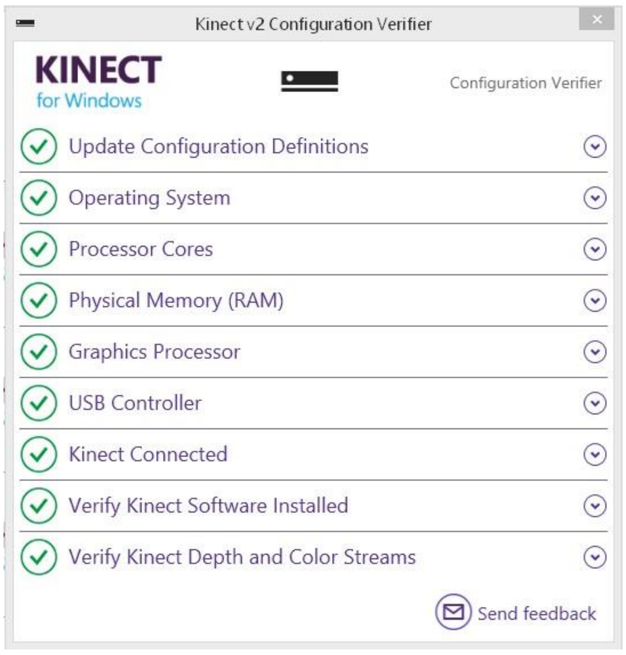
When developing this application Kinect for Xbox One sensor was used with the Kinect for Windows adapter. Writing the program was done with the latest Kinect for Windows SDK at the time – v2.0.

Connecting to the Kinect

After installing the Kinect for Windows SDK 2.0 the sensor bar was connected to the PC via an USB connection. The very first time the sensor is connected to a Windows computer, it will automatically install all required drivers.

Testing the Kinect compatibility

The Kinect for Windows SDK is provided with sample applications that can be used to check if the sensors video and infrared cameras are working properly and to demonstrate the body tracking abilities. A good way to know if the sensor is compatible with the computer that is used, would be to run the Kinect Configuration Verifier in the SDK Browser (Kinect for Windows) v2.0. It analyzes the computer for some hardware incompatibilities and verifies communication with the sensor (Figure 7).


Kinect v2 Configuration Verifier

KINECT for Windows

Configuration Verifier

- ✓ Update Configuration Definitions
- ✓ Operating System
- ✓ Processor Cores
- ✓ Physical Memory (RAM)
- ✓ Graphics Processor
- ✓ USB Controller
- ✓ Kinect Connected
- ✓ Verify Kinect Software Installed
- ✓ Verify Kinect Depth and Color Streams

Send feedback

Figure 7: Snippet of Kinect Configuration Verifier with used computer

Programming language

This application is developed using C# and built with a WPF (Windows Presentation Foundation) rendering engine in Microsoft Visual Studio 2014. This language was chosen because the author was most familiar with it and it is close to human readable code.

Programming applications with the SDK 2.0

When the sensor has been connected and verified, the first step would be to create a new Visual Studio project and to choose Visual C# from installed templates. Next the WPF application should be selected. After that the DLL (dynamic link library) should be referenced. It is provided by the Kinect for Windows SDK – Microsoft.Kinect.dll. This assembly can be found in the SDK installation directory (normally in C:\Program Files\Microsoft SDKs\Kinect\v2.0\Assemblies).

To enable handling the Kinect SDK reference the using directives should be added to the program:

```
using Microsoft.Kinect;
```

A new instance of the KinectSensor class, that is part of the SDK libraries, must be created and then used to control the device and read its information. Defining a Kinect sensor works the same as defining other class objects:

```
KinectSensor kinectSensor;
```

In this application instantiating the sensor is done in the Window_Loaded event. Method for getting default sensor object:

```
this.kinectSensor = KinectSensor.GetDefault();
```

Once the sensor object is retrieved, the method for starting the device must be invoked so that it can read from itself:

```
this.kinectSensor.Open();
```

Data frames

A frame contains data delivered from the sensor and each of them stores frame data temporarily to avoid memory allocation. Data should be retrieved from the frame and disposed as soon as possible to free up memory. It is possible to get access to six individual frames: color, depth, body, body index, infrared and long exposure infrared. For this application access to the infrared, depth and body data is required.

Infrared frame

Provides a black and white scene, but is actively lit, so brightness is consistent regardless of location and amount of light. Infrared is derived from the same sensor as depth. The infrared frames are stored as 16-bit unsigned integers and used for facial recognition, green screening, tracking reflective markers and filtering out jittery depth pixels.

To get infrared frames a reference should be retrieved for the frame:

```
InfraredFrameReader  infraredFrameReader;

this.infraredFrameReader  = this.kinectSensor.InfraredFrameSource.OpenReader();
```

A FrameArrived event fires every time a frame of the specified type is available and each time a method will be invoked (Reader_InfraredFrameArrived) where the raw data is processed:

```
this.infraredFrameReader.FrameArrived += this.Reader_InfraredFrameArrived;
```

When processing the data the actual infrared frame needs to be acquired by using AcquireFrame(). Then a buffer size is created for the incoming infrared frame and CopyFrameDataToArray() is used to copy the byte array of pixel data to a newly created buffer.

Depth frame

Also stored as a 16-bit unsigned integer where each value is in millimeters. Each pixel represents distance of closest object seen by that pixel. Used to build custom tracking algorithms.

Retrieving and processing depth frames works the same way as with infrared data.

Body frame

Contains computed information of people in the field of view of the sensor that include hand states, skeletal joints and orientations. [34]

Firstly the reader for body frames must be opened and a frame arrived event listener wired to it:

```
BodyFrameReader bodyFrameReader;
this.bodyFrameReader = this.kinectSensor.BodyFrameSource.OpenReader();
this.bodyFrameReader.FrameArrived += this.Reader_FrameArrived;
```

After retrieving the body frame with AcquireFrame(), it is copied to an array for body objects using GetAndRefreshBodyData() method. The first time this method is called, the sensor will allocate each body in the array and these body objects will be re-used as long as they are not disposed or set to null.

Cursor movement

In this application the wrist joint coordinates are tracked in order to move the cursor. Initially the hand joint coordinates were used, but because the hand moves too much during hand pose changes, the wrist is a more stable joint to track.

Acquiring joint coordinates

Checking if joint is tracked:

```
body.Joints[JointType.WristRight].TrackingState == TrackingState.Tracked
```

Initiating joint to be tracked:

```
Joint jointRightWrist = body.Joints[JointType.WristRight];
```

The position of the wrist must be scaled to the screen size so that moving the cursor would be more natural. For this the SystemParameters.PrimaryScreenWidth and SystemParameters.PrimaryScreenHeight are used to get the height and width of the primary monitor in pixels and position values that determine distance that the hand can move along the x and y axis.

```
Joint scaledRightWrist = Helpers.ScaleTo(jointRightWrist,
(int)SystemParameters.PrimaryScreenWidth,
(int)SystemParameters.PrimaryScreenHeight, SkeletonMaxX, SkeletonMaxY);
```

Initialization gesture

For this application to assign the cursor to the user’s hand, an engagement gesture will have to be performed by the user. There are two ways of teaching a machine to detect custom gestures using a Kinect sensor. The heuristic method depends on coding. This means that all the phases of the gesture must be checked programmatically. For example determining if a hand is above the head only the head and hand coordinates have to be compared. This method is used in this interface for determining hand poses. For more complex gestures the heuristic method requires a very high level of skill in programming and understanding of the human body.

The second method – machine learning is done by using the Visual Gesture Builder, which is a part of the Kinect for Windows SDK v2.0. This tool uses algorithms to understand data given as clips recorded with the Kinect sensor. In this project machine learning was used to teach the program to detect the engagement gesture. The gesture is moving the right hand up next to the shoulder with the elbow bent and the palm of the hand facing the sensor (Figure 8).

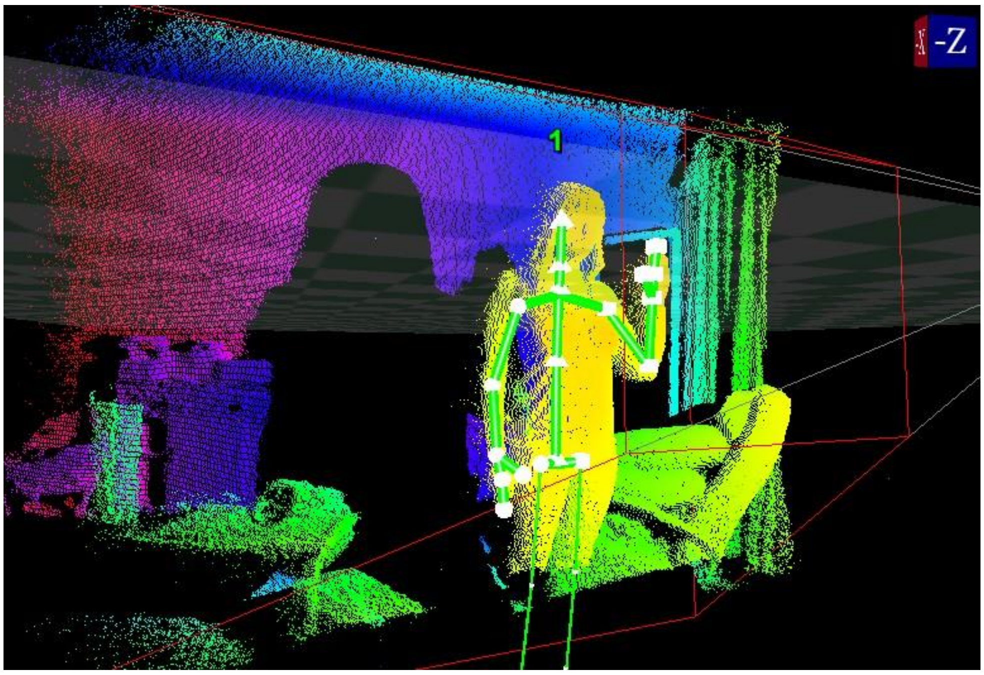
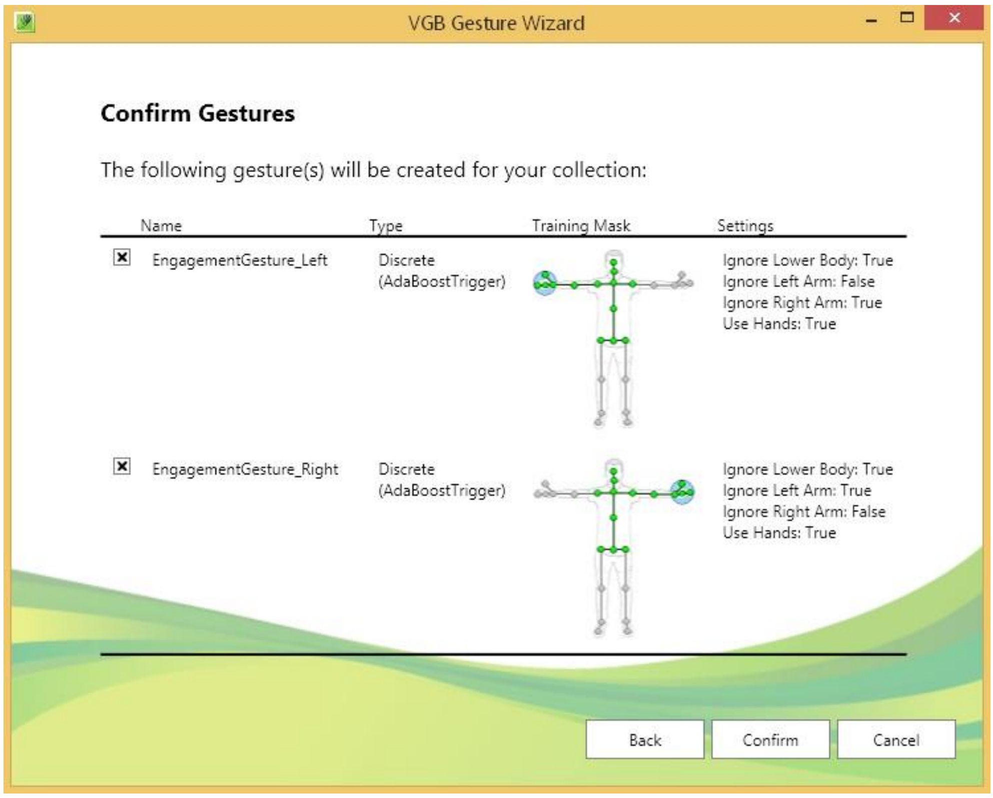


Figure 8: Engagement gesture that will assign the cursor to hand coordinates

The first step to train a gesture is recording clips with Kinect Studio, which is also a part of the SDK. To record a clip it is required to connect to the Kinect sensor and choose the streams that will be used in the recording. For this gesture only the body, infrared and depth streams were needed. Then the user will click the “Start recording file” button and stand in front of the sensor to act out the gesture they want to teach the computer and then stop recording. The more clips recorded of the gesture, the more accurate the gesture detection will be. In this case the engagement gesture was recorded four times for each hand and for testing two clips were recorded for each hand.

The next step is starting up Visual Gesture Builder and under the “File” tab choosing “New solution”. After the solution has been created a wizard can be used to create projects in the solution (right click on solution name and choose “Create new project with wizard”). In the wizard the name of the gesture will be asked. In this case it was named EngagementGesture. Next it is determined if the lower body joints (below the hips) will be used when building the gesture. For this gesture they are not used. It is also resolved if the hand state should be monitored during the action. At the end of this gesture the hand should be open. Next the wizard will need to determine which joint mask will represent the gesture. This shows which joints will be tracked (green) and which ones will not (grey) (Figure 9). After that it is determined if it is important to know which side of the body the gesture responds to. For this action it is needed to know if the right or left arm is doing the gesture. It is also possible to duplicate and mirror data. This is useful if motions for the left and right side of the body are equivalent. For this gesture it was set to yes, because the application should be able to track the right and the left hand as the hand will tire during long time use. The wizard will check if it is needed to detect progress during this gesture. This means it will be tracked how much of the gesture has been completed. For the engagement motion it is only needed to know if the gesture has been completed (true or false), so it was set to no. Next the overview of the selections for the gesture will be shown and have to be confirmed (Figure 9).



Confirm Gestures

The following gesture(s) will be created for your collection:

	Name	Type	Training Mask	Settings
☒	EngagementGesture_Left	Discrete (AdaBoostTrigger)		Ignore Lower Body: True Ignore Left Arm: False Ignore Right Arm: True Use Hands: True
☒	EngagementGesture_Right	Discrete (AdaBoostTrigger)		Ignore Lower Body: True Ignore Left Arm: True Ignore Right Arm: False Use Hands: True

Figure 9: Overview of the gesture created with Gesture Wizard

In this case the result will be four projects - two for each side of the body. One of the projects is for testing (.a) and the other one is for training. The recorded clips must be added to the projects (right click on project and “Add clip”) and the clips for training should be tagged. Tagging means going through the clip frame by frame and defining if the gesture is true during the current frame. All other frames are considered as false, but can be tagged as false manually if needed. While tagging the view of the infrared and body are shown, so it will be easy to determine the body position in a specific frame. After all of the clips have been tagged and the changes saved the detector must be built to get a database file (right click on solution, choose “Build”).

This database file (EngagementGesture.gbd) can be quickly tested live by clicking “File”, “Live preview…” and running the application with the resulting database file. This will connect to the sensor and the user can stand in front of it and reproduce the gesture that has to be detected. An output will show the level of confidence as bars. The higher the higher the confidence is, the more certain the machine is that the gesture is actually happening. With this gesture the resulting confidence was very high – up to a hundred percent.

To add this custom gesture recognition to the solution the database file must be added to the project as a content item. For that a new folder was created called Database and it was copied there. The “Copy to output directory” property (right click, “Properties) of the file must be changed to “Copy always”. Next the Microsoft’s Visual Gesture Builder library must be referenced which utilizes the Visual Gesture Builder APIs (application programming interface). [35]

To poll from the gesture database the path to it must be set:

```
        private readonly string gestureDatabase = @"Database\EngagementGesture.gbd";
```

And the name of the discrete gesture in the database:

```
        private readonly string engagementRight = "Engagement_Right";
```

Next the Visual Gesture Builder frame source should be created with the active sensor and the body index is initially set to 0, because that will be received from body source frame. Also the frame source reader will be opened:

```
       vgbFrameSource = new VisualGestureBuilderFrameSource(kinectSensor, 0);
       vgbFrameReader = vgbFrameSource.OpenReader();
```

The frame arrived event listener will be wired to the frame reader:

```
       vgbFrameReader.FrameArrived += Reader_FrameArrived;
```

The database is opened and the gesture needed is added to the gesture frame source:

```
    vgbFrameSource.AddGesture(gesture);
```

In the method invoked when the frame has arrived the frame is acquired and checked if it is not empty. Then the discrete gesture results which arrived with the frame must be acquired:

```
    IReadOnlyDictionary<Gesture, DiscreteGestureResult> discreteResults = frame.Dis-
    creteGestureResults;
```

The check must be performed if the gesture name received is the same as the one that is needed and an attempt to get value of the gesture is made:

```
    discreteResults.TryGetValue(gesture, out result);
```

This is used to find out if the gesture has been detected and how high is the confidence (a value from (0-1). If it has been detected in the frame and the confidence is more than 0.8 then a Boolean value `engagementGestureRecognized` is set to true, the frame reader will be paused to save resources and the cursor will be assigned to the hand that repeated the gesture. Disengagement occurs when user leaves the field of view of the sensor.

Assigning cursor to hand coordinates

The scaled coordinates of the tracked wrist will be sent to the Windows OS (operating system) and mouse movement will be simulated when user moves the tracked hand. For this to be possible the Microsoft Windows library file user32.dll must be imported. Programs call functions from Windows USER to among other things receiving windows messages like mouse and keyboard events. [36] [37]

Mouse event flags:

```
    public const int MouseEventMove = 0x01;
    public const int MouseEventLeftDown = 0x02;
    public const int MouseEventLeftUp = 0x04;
    public const int MouseEventRightDown = 0x08;
    public const int MouseEventRightUp = 0x10;
    public const int MouseEventAbsolute = 0x8000;
    public const int MouseEventWheel = 0x0800;
    public const int MouseEventHWheel = 0x01000;
```

The MouseEventMove flag is raised and as a result the mouse movement is simulated. Sending mouse event inputs:

```
Helpers.SendMouseInputMove(cursorX, cursorY,
(int)SystemParameters.PrimaryScreenWidth,
(int)SystemParameters.PrimaryScreenHeight);
```

The cursor movement with the application is smooth and intuitive, but while tracking the hand joint confusion may occur when certain joints are in the same position. This causes the mouse cursor to "jump" from one point to another. For example when the user is holding the tracked hand in front of the body and other joints of the body near the hand will be mistaken for this it.

Metrilus Aiolos library

Metrilus is a company in Germany that offers consultations and custom solutions for real-time 3D applications using various sensor types. [38]

The finger tracking library from Metrilus, Aiolos for Kinect v2, works coupled with the Kinect for Windows SDK v2.0. It relies on the body tracking provided by the SDK. The hand body joint is used for initialization. A patch around this point is cut out of the distance and the size of the patch is dependent on the distance from the camera. Next the hand is separated from the background. Based on a mask image generated by the foreground image (hand only) a skeleton is computed. The skeleton graph is analyzed and the corresponding finger tips and middle and root joint positions are found.

Features:

- 2D and 3D positions of the fingers tip-, middle- and root joints
- 2D and 3D contour points of the hand
- Finger labeling (experimental) [39]

In March 2015 Aiolos was released under a Creative Commons license and is free to use for non-commercial purposes. [40] The library can be downloaded from the Metrilus homepage. Aiolos must be referenced in the project and the using directives added to the program:

```
using Metrilus.Aiolos;
```

To acquire fingers joint data using Aiolos library the display width and height, infrared data, depth data, skeleton data and coordinate mapping have to be passed:

```
KinectHand[] hands = engine.DetectFingerJoints(width, height, irFrameData,
depthFrameData, bodies, coordinateMapper);
```

Because the Kinect color camera, depth sensor and infrared sensor have different resolutions, the fields of view also differ. The CoordinateMapper is used to convert 3D skeleton joint positions to 2D color or depth space points and the other way around. [30] [39]

Mouse events

Mouse events are triggered by different hand poses identified with the help of the Metrilus Aiolos library. The pictures of how mouse events can be controlled with the hand in this application are taken as snippets using the Aiolos demo program.

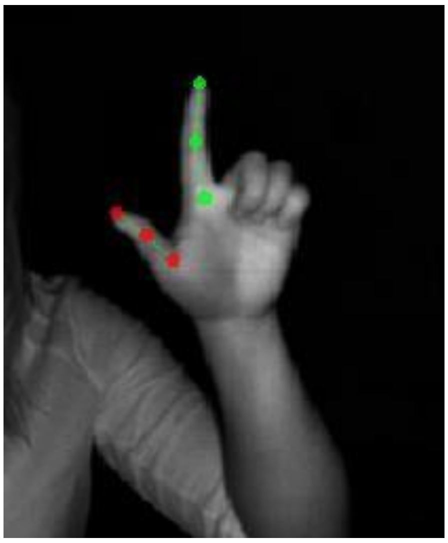


Figure 10: Left click

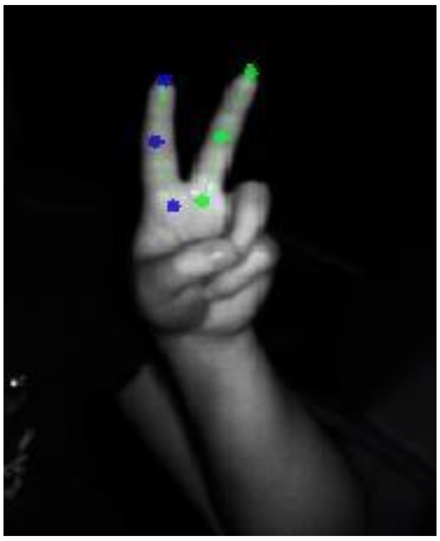


Figure 11: Double click

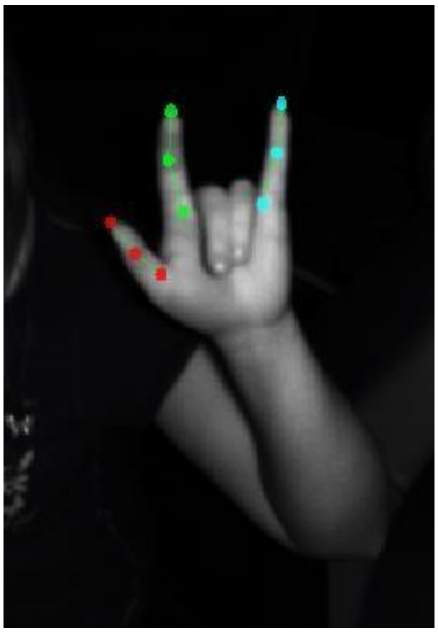


Figure 12: Right click

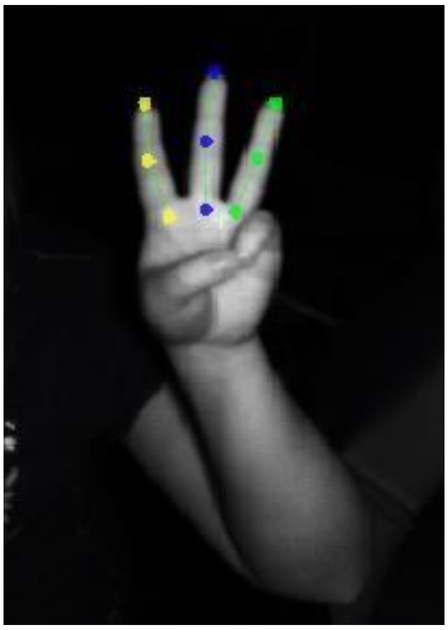


Figure 13: Scroll down

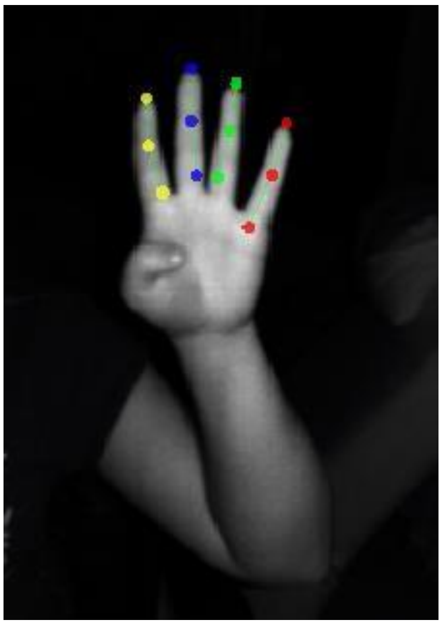


Figure 14: Scroll up

Problems with the library:

- A single finger is always tracked as an index finger
- When all fingers except the thumb are tracked, it apprehends the hand as though the palm is directed away from the sensor (same with tracking only middle three fingers)
- It can only track fingers if they are separated from each other
- When the hand is sideways and the palm is not facing the sensor properly, the library can mistake it with the thumb and index finger being seen. This causes false left clicks

Threading is used to avoid delays in moving the cursor. Tracking the wrist joint and assigning it as the cursor is done in one thread and the finger tracking library and other mouse event handling runs in the second thread. Determining if a hand gesture has been made and mouse events happen with minimal delay with this application, which is a very positive feature.

Simplified flow chart of the application

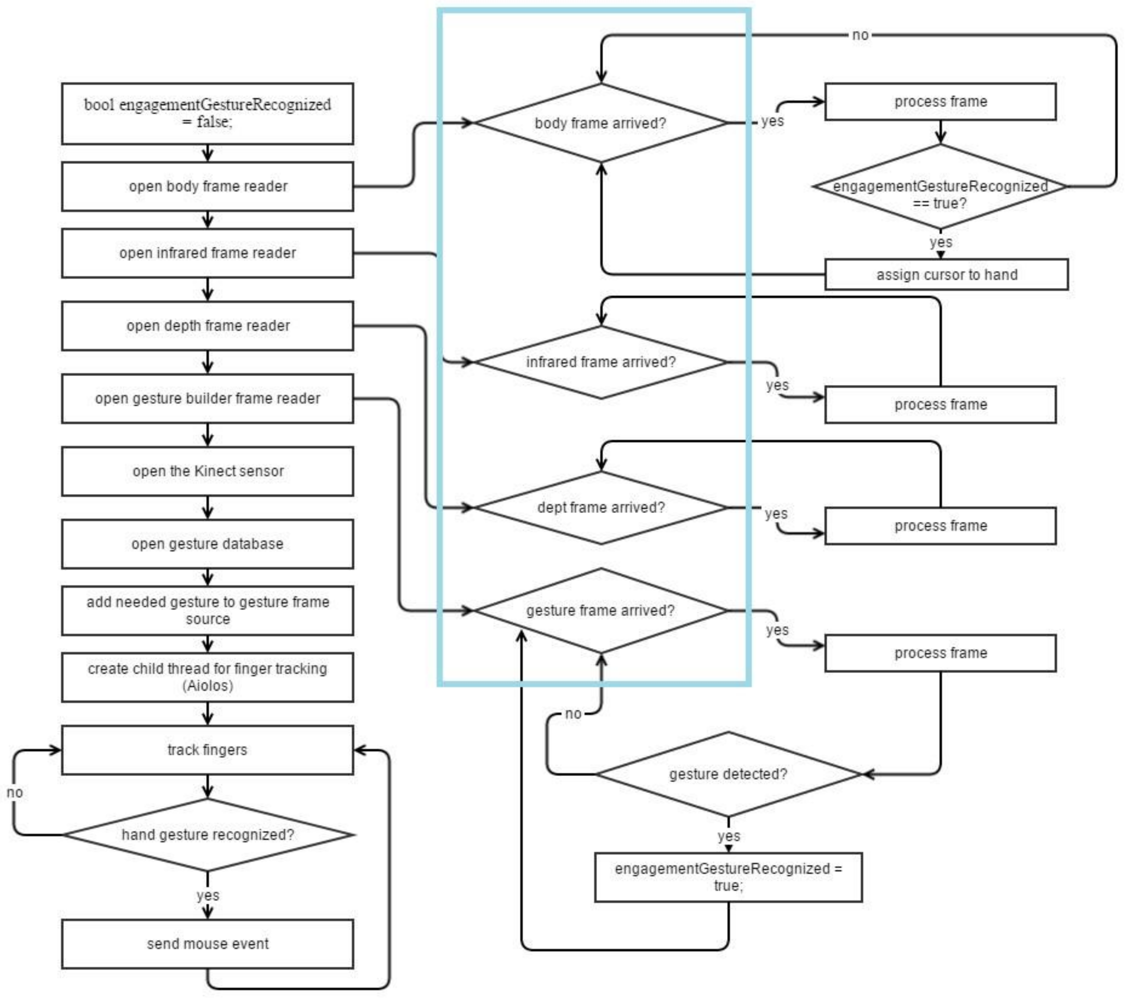


Figure 15: Flow chart to give a simplified overview of what happens in the code (blue box marks the event listeners)

Running the application

The requirements for using this application is to have the Kinect for Windows SDK v2.0 and .NET Framework 4.5 installed on the computer that is used. The computer must also have 64-bit Windows 8 or 8.1 installed and be equipped with a Kinect for Xbox One or Kinect for Windows v2 and the drivers for the device must be installed. When the requirements have been met, the application can be run by double clicking on the .exe file .(…\KinectCursor\bin\x86\Debug\KinectCursor.exe).

Comparison to other similar solutions

A master’s thesis topic in 2012 was creating an interface which uses the Kinect sensor v1 to control a Windows 7 mouse with using only gestures. At that time the newest version of Kinect for Windows SDK was Beta 1. The application uses the same method to control the mouse cursor as in this project. The biggest difference is that finger detection is not used and mouse left click occurs when the hand has not been moved (and with it the cursor) for 2 seconds. The other mouse actions can be triggered when after the left click the hand stays immobile for 3 more seconds and then a menu window appears where double click, right click or drag can be chosen. If the hand moves even a little bit, the action will not succeed. [40] In the authors opinion this solution is not really user friendly and takes a lot of time to navigate. The hand will tire faster when you have to hold it up and be immobile. Then again at that time the first sensor was not as advanced as the new version and the possibilities were more limited.

Another similar application that was found also used the same method to assign the cursor movement to the hand movement, but it only implements the mouse left click event. This is done by raising the left hand. In the documentation there is not mentioned how far up the hand must be lifted and the author of this thesis was not able to run and test the application because the first version of the Kinect sensor was used when developing it and the applications created for that sensor are not compatible with the new device. [41]

Conclusion

The main goal of the bachelor's thesis was to develop an application that allows users to interact with a computer without using any intermediate devices that require physical contact. The resulting interface makes it possible to use the functionalities of a Windows desktop relying only on gestures, provided the computer meets the requirements of using a Kinect sensor and is equipped with one. Because the library used to track fingers is still in development, the functionalities are limited at this point. The possibilities for future development in human-computer interaction using vision based hand recognition are endless and the interaction will become more natural and effortless. As the equipment used to recognize hand poses is becoming more affordable and available, the possibility that mechanical devices which need physical contact will become obsolete becomes progressively more of a reality.

Kokkuvõte

Žestipõhine arvuti kontrollimine kasutades Kinect sensorit

Bakalaureusetöö põhieesmärgiks oli välja töötada rakendus, mis võimaldab kasutajatel suhelda arvutiga kasutamata vahepealseid seadmeid, mis vajavad füüsilist kontakti. Tulemuseks olev liides teeb võimalikuks kasutada Windows töölauda toetudes vaid käeliigutustel tingimusel, et arvuti täidab nõudeid, mis on vajalikud Kinect sensori kasutamiseks ja on sellega ka varustatud. Kuna teek, mida kasutatakse sõrmede äratundmiseks on veel arendusfaasis, on hetkel selle funktsionaalsus piiratud. Võimalused edasiseks arenguks inimese ja arvuti vahelises interaktsioonis kasutades visuaalset käe eristamist on lõputud ja suhtlus muutub järjest loomulikumaks ja vähem vaevanõudvaks. Kuna seadmed, mida kasutatakse käe pooside äratundmiseks, muutuvad järjest taskukohasemaks ja kättesaadavateks, siis võimalus, et mehhaanilised seadmed, mis vajavad füüsilist kontakti, muutuvad iganenuks, kujuneb järjest reaalsemaks.

References

1. Wikipedia, Kinect, http://en.wikipedia.org/wiki/Kinect (25.04.2015).
2. Dutta, T. (2012). Evaluation of the Kinect™ sensor for 3-D kinematic measurement in the workplace. Applied ergonomics, 43(4), 645-649
3. Business Insider, The Story Behind Kinect, Microsoft’s Newest Billion Dollar Business, http://www.businessinsider.com/the-story-behind-microsofts-hot-selling-kinect-2011-1?op=1 (25.04.2015).
4. Guinness World Records, Fastest-selling gaming peripheral, http://www.guinnessworldrecords.com/world-records/fastest-selling-gaming-peripheral/ (25.04.2015).
5. Abhijit Jana, Kinect for Windows SDK Programming Guide, Packt Publishing, Birmingham –Mumbai, 2012.
6. Microsoft homepage, Kinect for Windows Sensor Components and Specifications, https://msdn.microsoft.com/en-us/library/jj131033.aspx (05.05.2015).
7. Roland Smeenk, Kinect v1 and Kinect v2 fields of view compared, http://smeenk.com/kinect-field-of-view-comparison/ (05.05.2015)
8. Channel 9, Kinect 1 vs. Kinect 2, a quick side-by-side reference, http://channel9.msdn.com/coding4fun/kinect/Kinect-1-vs-Kinect-2-a-side-by-side-reference (05.05.2015)
9. Wikipedia, Kinect for Xbox One, http://en.wikipedia.org/wiki/Kinect_for_Xbox_One (05.05.2015)
10. Microsoft homepage, Kinect for Windows features, https://www.microsoft.com/en-us/kinectforwindows/meetkinect/features.aspx (05.05.2015)
11. Webb, J., & Ashley, J. (2012). Beginning Kinect Programming with the Microsoft Kinect SDK. Apress
12. Kinect for Windows Blog, Microsoft to consolidate the Kinect for Windows experience around a single sensor, http://blogs.msdn.com/b/kinectforwindows/archive/2015/04/02/microsoft-to-consolidate-the-kinect-for-windows-experience-around-a-single-sensor.aspx (05.05.2015)
13. Wikipedia, Image sensor, http://en.wikipedia.org/wiki/Image_sensor#CCD_vs_CMOS (08.05.2015)

14. John MacCormick, How does the Kinect work? , http://www.cs.bham.ac.uk/~vvk201/Teach/Graphics/kinect.pdf (08.05.2015)
15. Butkiewicz, T. (2014, September). Low-cost coastal mapping using Kinect v2 time-of-flight cameras. In Oceans-St. John's, 2014 (pp. 1-9). IEEE
16. Popular Science, Exclusive: Inside Project Natal's Brain (The Artificial Intelligence Behind Microsoft's Xbox 360 Motion-Sensing Game Controller), http://www.popsci.com/gadgets/article/2010-01/exclusive-inside-microsofts-project-natal (07.05.2015)
17. Wikipedia, Time-of-flight camera, http://en.wikipedia.org/wiki/Time-of-flight_camera (8.05.2015)
18. Wikipedia, Photon, http://en.wikipedia.org/wiki/Photon (10.05.2015)
19. Dal Mutto, C., Zanuttigh, P., & Cortelazzo, G. M. (2012). Time-of-flight cameras and microsoft Kinect™. Springer Science & Business Media
20. Engadget, Microsoft's Project Natal roots revealed : 3DV Systems ZCam, http://www.engadget.com/2009/06/03/microsofts-project-natal-roots-revealed-3dv-systems-zcam/ (10.05.2015)
21. Kinect for Windows Blog, The Kinect for Windows v2 sensor and free SDK 2.0 public preview are here, http://blogs.msdn.com/b/kinectforwindows/archive/2014/07/15/the-kinect-for-windows-v2-sensor-and-free-sdk-preview-are-here.aspx (11.05.2015)
22. González-Ortega, D., Díaz-Pernas, F. J., Martínez-Zarzuela, M., & Antón-Rodríguez, M. (2014). A Kinect-based system for cognitive rehabilitation exercises monitoring. Computer methods and programs in biomedicine, 113(2), 620-631.
23. Lange, B., Chang, C. Y., Suma, E., Newman, B., Rizzo, A. S., & Bolas, M. (2011, August). Development and evaluation of low cost game-based balance rehabilitation tool using the Microsoft Kinect sensor. In Engineering in Medicine and Biology Society, EMBC, 2011 Annual International Conference of the IEEE (pp. 1831-1834). IEEE
24. Miles, R. (2012). Start Here! TM Learn the KinectTM API. "O'Reilly Media, Inc
25. Zhang, Z. (2012). Microsoft kinect sensor and its effect. MultiMedia, IEEE, 19(2), 4-10
26. Microsoft, Xbox One Kinect placement, https://support.xbox.com/en-US/xbox-one/kinect/positioning-kinect-sensor (11.05.2015)

27. Polygon, Microsoft details placement suggestions for new Kinect, http://www.polygon.com/2013/10/23/4948866/microsoft-details-placement-suggestions-for-new-kinect (11.05.2015)
28. Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., ... & Fitzgibbon, A. (2011, October). KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In Proceedings of the 24th annual ACM symposium on User interface software and technology (pp. 559-568). ACM
29. Microsoft Download Center, Kinect for Windows SDK v1.8, https://www.microsoft.com/en-us/download/details.aspx?id=40278 (12.05.2015)
30. Microsoft, Set up the hardware, https://www.microsoft.com/en-us/kinectforwindows/purchase/sensor_setup.aspx (12.05.2015)
31. Miles, R. (2012). Using Kinect for Windows with XNA. Kinect for Windows SDK.
32. Microsoft Developer Network, What's New in the October 2014 Kinect for Windows version 2.0 SDK, https://msdn.microsoft.com/en-us/library/dn782041.aspx
33. Microsoft Developer Network, Kinect API Overview, https://msdn.microsoft.com/en-us/library/dn782033.aspx (16.05.2015)
34. Pinvoke, mouse_event (user32), http://pinvoke.net/default.aspx/user32.mouse_event (16.05.2015)
35. Wikipedia, Microsoft Windows library files, http://en.wikipedia.org/wiki/Microsoft_Windows_library_files#USER32.DLL (16.05.2015)
36. Channel 9, Custom Gestures End to End with Kinect an Visual Gesture Builder, http://channel9.msdn.com/Blogs/k4wdev/Custom-Gestures-End-to-End-with-Kinect-and-Visual-Gesture-Builder#time=00h02m23s (20.05.2015)
37. Metrilus GmbH, Company, http://www.metrilus.de/company/ (17.05.2015)
38. Metrilus GmbH, Metrilus Aiolos Finger Tracking, http://www.metrilus.de/blog/portfolio-items/aiolos/ (17.05.2015)
39. Metrilus GmbH, Aiolos released under CC license, http://www.metrilus.de/blog/aiolos-released-under-cc-license/ (17.05.2015)
40. CodeProject, Understanding Kinect Coordinate Mapping, http://www.codeproject.com/Articles/769608/Understanding-Kinect-Coordinate-Mapping (17.05.2015)
41. CodePlex, Kinect Mouse Cursor, https://kinectmouse.codeplex.com/ (20.05.2015)