Natural user interfaces from all angles

An investigation of interaction methods

using depth sensing cameras

A Medialogy project by

Michael Birkehøj Jensen

AALBORG UNIVERSITET

Project title:

Natural user interfaces from all angles

Department of Architecture,
Design and Media Technology

Medialogy, 10th Semester

Project Period:

8-2-2011 / 31-5-2011

Semester Theme:

Master Thesis

Supervisor:

Mads Græsbøl Christensen

Projectgroup no.:

mea111041

By:

Michael Birkehøj Jensen

Abstract:

With the emergence of natural user interfaces a new task of designing meaning full interaction has come along. Would it make sense to not have to get up to get the remote control, if you had to get up anyway to interact with an interface controlled by a camera... This project investigated the possibilities of natural interfaces using depth sensing camera technology to control a media center like interface. The project proposes new methods of interaction using only the body as a controller, which supposedly should perform better at all angles to the interface. This means that the user would not have to get up, but would be able to control the interface right away from the initial position.

To test the hypothesis, the project designs and implements a test interface set in the scene of a typical media center interface. The test users are presented with tasks to select different movies by moving the cursor to the right movie box cover and holding it there fore a little while. This same task of selecting the right movie is iterated with different system configuration of user position relative to the interface difficulty and interaction method.

The project found that there is a way for the proposed methods, however more accurate user tracking in a larger area is needed for the proposed interaction methods will be competitive to traditional remote control.

Printed copies: . . . . . 2.

Appendix: . . . . . Included in the back of the report

. . . . . CD can be found with the report

Table of content

1. Preface 1

2. Introduction 3

2.1. Delimitations 5

2.2. Readers guide 5

3. Analysis 7

3.1. Interaction and natural interfaces 9

3.2. Interaction 10
3.2.1. Depth sensing cameras 10
3.2.2. Analysis of depth information 11

3.3. Graphics and visualization 12

3.4. Tools for implementation 12
3.4.1. Damping and noise filtering 13

3.5. OpenNI 13
3.5.1. Production Nodes 14
3.5.2. Production chains 15
3.5.3. Capabilities 15
3.5.4. Generating and Reading Data 15
3.5.5. Main Objects 16
3.5.6. Configuration Using XML 18

3.6. Conclusion of analysis 19

4. Design 21

4.1. Interface design and graphics 23
4.1.1. The test interface 23
4.1.2. Buttons and interactive elements. 26
4.1.3. Graphics design 28

4.2. Interaction method design 30
4.2.1. Mouse or track pad 31
4.2.2. Hand direct. 31
4.2.3. Line of sight 32
4.2.4. User reach area mapping 32

4.3. Software design 34

4.4. Sound design. 34

4.5. Conclusion of design. 34

5. Implementation 37
5.1. Hardware setup 39
5.2. Software implementation 39
5.2.1. OpenNI and Nite. 39
5.2.2. GUI implementation 41
5.2.3. Positioning the test interface GUI 42
5.2.4. Calculation and animation of wait timer bar 43
5.2.5. Consistency of the users experience. 43
5.2.6. Sound implementation 43
5.2.7. Implementation of interaction methods 43
5.3. Conclusion of implementation. 46
6. Test. 47
6.1. Test setup 49
6.1.1. Actual test 49
6.1.2. Test database and data gathering 50
6.2. Results. 51
6.3. Data processing analysis 51
6.3.1. Initial data processing. 52
6.3.2. Test material 53
6.3.3. Classifying data 55
6.3.4. Qualitative results 64
7. Evaluation. 67
8. References 71

Table of figures

Figure 1. The concept of "Flow" [http://amandakatarina.files.wordpress.com/2010/04/1000px-challenge_vs_skill-svg4.png] 10
Figure 2. System overview of the Primesense depth camera technology [http://www.primesense.com/?p=514] 11
Figure 3. Line plane intersection - [http://en.wikipedia.org/wiki/Line-plane_intersection] 13
Figure 4. Xbox Zune user interface 24
Figure 5. Xbox kinect user interface 24
Figure 6. Xbox Zune user interface 25
Figure 7. First implementation of the test interface. 25
Figure 8. Wireframe interface mockup. 2 by 2 grid size. blue: button areas, yellow: main area.. 26
Figure 9. The three different states of the gui elements. Left: non-active icon, middle: activated icon, right: activated icon with interaction time indicator. 27
Figure 10. The cursor designed for the test interface 28
Figure 11. Graphical visualization of the found user position, with five different areas of angles to the interface. 28
Figure 12. Simple one line movie icon interface design 29
Figure 13. Large icons on the movie selector interface. 30
Figure 14. Small icons on the movie selector interface. 30
Figure 15. Direct hand position mapping concept. 31
Figure 16. Concept setup of line of sight method. 32
Figure 17. User reach area mapping interaction method concept diagram 33
Figure 18. Mapping of user reach area to interface screen area 33
Figure 19. Concept setup of line of sight method, camera tilted relative to screen.. 45
Figure 20. Raw data - time sample histogram - 1000 bins 53
Figure 21. Histogram - time data - all samples - threshold 25 second - 50 bins 53
Figure 22. Relation between evaluators and information gain [http://www.useit.com/papers/heuristic/heuristic_evaluation.html]. 54
Figure 23. Standard deviation of N number of test users (Time measurement). 54
Figure 24. Standard deviation of error rate according to number of expert test users 55
Figure 25. Average standard deviation of user error rate according to number of expert users (10,000 random iterations) 55
Figure 26. Example of plotting data using PRTools - Success according to x- and y-position ... 56
Figure 27. Error rate for each interaction method, all test users included 57
Figure 28. User position with respect to class (interaction method) 58
Figure 29. Time statistics for each interaction method 59
Figure 30. X-position statistics for each interaction method. 59
Figure 31. Y-position statistics for each interaction method 60
Figure 32. 2D real world user position scatter plot with color indicated position difficulty. 61
Figure 33. Bar chart - Error rate for all combination of position and interface difficulty (mouse) 62
Figure 34. Bar chart - Error rate for all combination of position and interface difficulty (line of sight). 63
Figure 35. Bar chart - Error rate for all combination of position and interface difficulty (hand direct) 63
Figure 36. Bar chart - Error rate for all combination of position and interface difficulty (user area mapping). 64

Code examples

Code example 1. OpenNi XML configuration. 19
Code example 2. Initialization function called once when the context is created 40
Code example 3. Deconstructor for the OpenNIContext class. 40
Code example 4. InitThread, responsible for setting up the connection to OpenNI and the Nite Middleware 41
Code example 5. OpenNIContext class, Start()- and Update()-function 41
Code example 6. The OpenNI nodes used by the implementation 41
Code example 7. C# - unity gui transformation matrix. 42
Code example 8. C# - Unity gui coordinates. 42
Code example 9. C# - Button area size calculations pseudo code 42
Code example 10. Calculation of masking area effect for wait for interaction timer visualization 43
Code example 11. Matlab - classifying data according to position difficulty 60
Code example 12. Matlab - scaling time according to error rate. 62

List of tables

Table 1. http://www.primesense.com/?p=514 14
Table 2. Database design 1. for test data gathering. 44
Table 3. Database design 2. for test data gathering. 45
Table 4. Error correction test 1. Keyboard layout, test words and features 46
Table 5. Database design. 47
Table 6. Error rate for all combinations of interface- and position difficulty 55

Abbreviations and Definitions

GUI: Graphical User Interface

GUI layout:. . Sizing and positioning GUI elements to form a functional, visually attractive screen.

NUI: Natural user interface

IDE: Integrated Development Environment; a software development tool that includes at least an editor, a compiler and a debugger.

lo-fi: Abstract, low level of detail, visually imperfect

Hi-fi: High level of detail, visually elaborate, looking like real

Mockup: A non-interactive, high-fidelity representation of a GUI

OOP: "Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represent an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships."

Round-trip engineering: A functionality of software development tools that provides generation of models from source code and generation of source code from models; this way, existing source code can be converted into a model, be subjected to software engineering methods and then be converted back.

UI: User Interface

UML: Unified Modeling Language

Wireframe interface: Computer-drawn, low-fidelity version of a GUI that is used in the early stages of GUI design.

DADIU: "Det Danske Akademi for Digital, Interaktiv Underholdning" - educates students in making computer games, and is an association of university and art schools throughout Denmark

ECTS: European Credit Transfer and Accumulation System

3D: three-dimensional

2D: two-dimensional

1. PREFACE

The project was done in a 20 ECTS period while the rest of the 30 ECTS semester period was used on DADIU. Information on the DADIU production can be found in a separate report.

2. INTRODUCTION

This project will investigate the navigation of natural interfaces using depth sensing camera technology. Different methods of interacting with an interface, using the body as a controller has become known through the emergence of natural interface like Xbox Kinect and Nintendo Wii etc. The possibility of navigating an interface using the line of sight between the users eye and hand, as if pointing to a distinct point on a screen, will be compared to different methods of mapping the position of the users hands to the screen space of the graphical user interface. The idea is enabled by the break through of depth sensing cameras now available to the consumer marked. The project uses the Xbox Kinect camera to determine not only the position of the users hand but also the users head and body center in 3D space.

Normally the kinect experience is a more direct mapping between the users body and the screen space. This means that the user interaction with the interface needs to be somewhat directly in front of the camera and the screen at a distance of approximately 2-3 meters. If the user wants to interact with the lower part of the screen, the user has to move the hand down while being aware of the cursors position on the screen. The hypothesis is that other possible methods of interaction will allow better Interaction at angles, while omitting or weakening the need for visual representation of the point of interaction on the screen while still being intuitive and making way for faster more natural interaction. This would help the user to stay in flow and keep focus on the task at hand. The goals for the project therefore is to design two new interaction method along with a believable interface and test environment to test the performance of the interaction methods. To sum of the goal for the interaction methods, the idea is to provide natural and intuitive interaction from all angles.

The above will be evaluated in chapter "7. Evaluation" on page 67.

2.1. Delimitations

Setup

- The project limits itself to a known setup, and can not directly be used in another setup. If the screen size or position along with the relative position of the camera changes the result will be different. However converting the system to another screen size and position is trivial.

Users

- Only one user can interact with the system at any given time. If more users are seen by the camera at any time, only one will be considered active and capable of interaction with the system.

Test platform

- MacBook Pro - OS X 10.6.7
- Microsoft Kinect sensor
- Philips 42 inch high definition TV
- The Unity game engine
- OpenNI and Nite

2.2. Readers guide

The report is divided into 8 main chapters.

Chapter structure

- Introduction
- Analysis
- Design
- Implementation
- Test and results
- Evaluation
- References
- Appendix

Graphical indication can be found in the page margin if further information relevant to the current section is available in the appendix or on the appended CD.

Examples of graphical indicators can be seen below:

Additional information on CD 

Additional information in appendix 

References

Referencing is done using the following notation [#], # being the number of the reference in the list. See "8. References" on page 71. This in-text referencing is chosen over Harvard in-text reference style also known as parenthetical referencing, due to the fact that the reference are not primarily books easy identified by author and date.

Figures, tables and code examples

Figure and Table references can be found in respective "Table of figures" and "List of tables". If no external reference exist, the material is custom made for this report only and most not be used without reference. A list of code examples and pseudo code can be found in "List of code examples".

Cross references

Cross references within the report is done as follows. See "2.2. Readers guide" on page 5.

3. ANALYSIS

This chapter contains an analysis of natural interfaces, and tool needed to implement and test the purposed. What is needed to achieve the goals and what will need to be investigated.

3.1. Interaction and natural interfaces

Interfaces comes in many forms, the latest being natural interfaces like Xbox Kinect and Nintendo wii, etc. Where the control scheme is more and more shifted toward being a metaphor for normal physical interaction. Natural interfaces is the newest buzz word when it comes to interfaces. Natural users interface or "NUI" covers the term of interfaces which enables the user to interact in a more intuitive and natural manner. Human computer interaction have become an every day occurrence, and over the last few years the term natural interfaces has emerged.

When interacting with a screen interface using a the Nintendo Wii-mote, the user points it at the point on the screen by moving and tilting the Wiimote. This interaction can give the feeling of "shooting from the hip" due to the difficulty of determining where on the screen the pointing line will actually hit. This is dealt with by using graphical indication of the point of interaction in the form of a hand cursor tinted with a color to distinguish between more than one user.

The kinect way of interaction has some of the same strengths and weaknesses as the Nintendo Wiimote. It is a more free and physical way of interaction and does not depend on controller, but does also require some form of visual indication. In some cases typically during flat 2D'ish menu interaction the is done by indicating the curser with different hand icons like known from the mouse on a regular desktop or laptop. In other cases and entire semi transparent avatar is shown on the screen following the movements on the users body, this is for instance used in the game "Kinect Adventure".

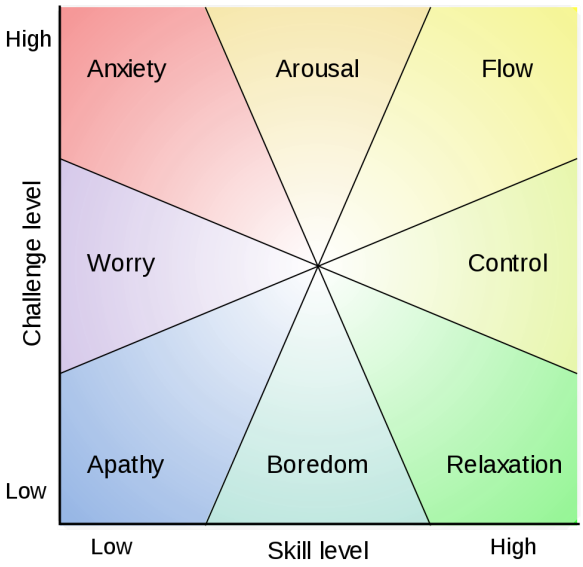
The two proposed interaction methods are "line of sight" and user area mapping. The idea for "line-of-sight" interaction originates from the human interaction of pointing. When pointing towards the arm of the person pointing acts as line towards what ever the person is pointing to with a reference to the users eyes. This could omit the need for semi transparent avatars or large cursors to indicate the current point of interaction on the screen. Typically giving the effect of extending the pointing arm towards the point of interest.

The "user area mapping" interaction, maps the area around the user to the screen as if the user were standing right in front of the screen scaled to a size fitting to the users area of reach.

User experience and interfaces

The term "flow" is often when designing computer games, but the general idea behind flow can be applied to a lot of different experiences from physical play, work environment and navigating an interface which is in its nature not very far from a simple computer game. The overall concept of flow can be seen in Figure 1.

Figure 1. The concept of "Flow" [http://amandakatarina.files.wordpress.com/2010/04/1000px-challenge_vs_skill-



svg4.png]

Among the eight main components of flow experience [2] is having a clear goal and getting clear feedback on whether the goal is getting closer. Another keypoint of flow is that the challenges matches the skills of the user, and that the user feels in control of the situation. A smooth interface which does what the user expects will increase the usability and enhance the entire experience.

To test the interaction methods in this context a interface and test environment must be established. To give the users a good experience of navigating an interface clear graphical indication of what is going on is needed. Graphical user interface or just GUI has to be developed to test the proposed interaction methods.

3.2. Interaction

To interact with a screen in the proposed manner, a method of reliably tracking the users movements is needed. Among technical possibilities is a variety of motion sensing systems which all have in common the the user must be have some kind of equipment, either in the for of a controller or tracking objects.

3.2.1. Depth sensing cameras

Another choice is using camera techniques to track users. Depth information can be found using either "time of flight" cameras [5] like D-IMager [6], stereo vision or camera systems using projection of structured light [7].

The Microsoft Kinect camera uses a version of structured light projection [4]. An overwiev of how it works can be seen in Figure 3.

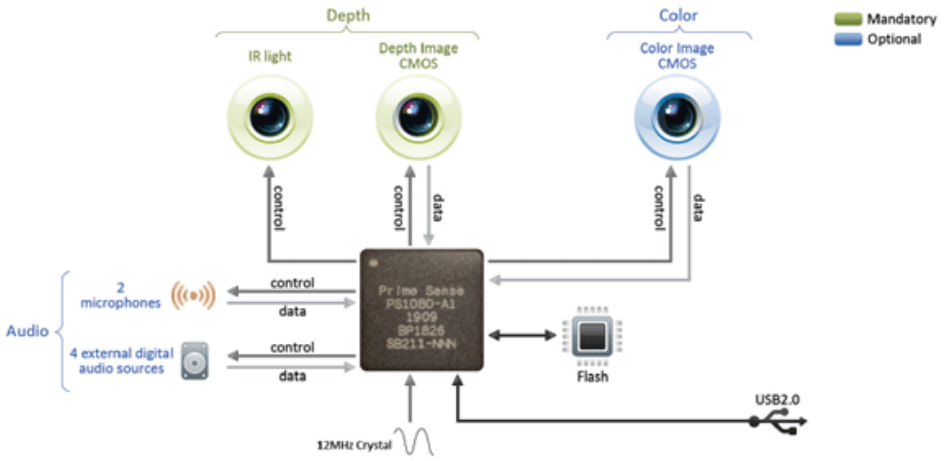


Figure 2. System overview of the Primesense depth camera technology [http://www.primesense.com/?p=514]

3.2.2. Analysis of depth information

Information about depth is not directly useful for interaction, however the information can be processed and analysed to segment out users, hands, faces and entire skeletons.

PrimeSense, OpenNI and NITE

The company behind the depth sensing technology used by the Microsoft Kinect camera, has released an open software package to enhance the evolvement of natural interfaces [15]. As mentioned it is possible to access Kinect camera data from within the unity game engine, but another advantage of using Unity is that it is possible to port OpenNI and Nite to work with Unity on Windows and with a bit more work also other platforms like Mac OS X, Linux etc. This is possible sinse the OpenNI core is written completely in C. OpenNI and Nite supplies much of the functionallity found in commercial products using the Kinect camera like for instance Xbox games like "Kinect Adventures". This means that by using OpenNI and Nite for this project will enable real comparesons with the current state of the art interfaces on the consumer marked.

Alternative to using OpenNI and Nite would be to implement the system using a image processing library like OpenCV [20]. Since the purpose of the project is to investigate the use of new interaction method and not to implement the a system to segment and track users from a depth image, utilizing a well tested library will be suitable.

3.3. Graphics and visualization

A quick view at easy accessible possibilities for implementing the graphical, reveals a variety of different possibilities each with pros and cons. Preferably both visualization and vector math calculation in one solution

Processing

Processing is a small application based on the Arduino programming environment [22], and uses a syntax similar to Java and C. [8]. Processing has the advantage of being easy accessible, along with already having been tested to work with the Microsoft Kinect camera [9].

Cinder

Cinder is a C++ library specifically tailored for creative application [10]. Cinder has the advantage of harnessing the power and speed of C++ and OpenGL accelerated graphics. However Cinder is less easy accessible than for instance processing a has a steeper leaning curve. Getting data from the Kinect camera into Cinder has been confirmed and tested [11].

Unity

As a development tool that has been designed to let users focus on creating amazing game, Unity has simplified the trivial technical tasks behind making 3D computer games. This being said unity can be used for much more than just creating games. Unity has the advantage of being able to harness the power of graphics acceleration utilized through high level scripting interface using either JavaScript or C#. Further more Unity comes with Nvidia PhysX engine build in and easy accessible from script [14]. Scripting is build upon the open source cross platform .NET development platform "Mono" [13], which gives the flexibility to use .NET libraries and dll files [12]. This means that Unity can access Kinect via dll files even on Mac OS X.

3.4. Tools for implementation

To implement the proposed interaction methods different math tools will be needed. For the line of sight methods creating a line in 3D space from two positions in space, along with the lines intersection with the plane where the screen is situated.

Line plane intersection

The proposed interaction method of using the users line of sight through the hands of the user, will need some calculation to convert the line of sight into a 2D point on the graphical interface. To locate the point where the user points in the 2D camera space or 3D line plane intersection is needed. Since the line is given by two points in space, respectively head and hand, and the plane can be described as a point and a normal vector for the plane at the position of the camera, the algebraic models become.

In vector notation, a plane can be expressed as

$$(\mathbf{p} - \mathbf{p_0}) \cdot \mathbf{n} = 0$$

where $\mathbf{n}$ is a normal vector to the plane and $\mathbf{p_0}$ is a point on the plane. The vector equation for a line is

$$\mathbf{p} = d\mathbf{l} + \mathbf{l_0}$$

where $\mathbf{l}$ is a vector in the direction of the line and $\mathbf{l_0}$ is a point on the line. Substitute the line into the plane equation to get

$$(d\mathbf{l} + \mathbf{l_0} - \mathbf{p_0}) \cdot \mathbf{n} = 0$$

Distribute $\mathbf{n}$ to get

$$d\mathbf{l} \cdot \mathbf{n} + (\mathbf{l_0} - \mathbf{p_0}) \cdot \mathbf{n} = 0$$

And solve for d

$$d = \frac{(\mathbf{p_0} - \mathbf{l_0}) \cdot \mathbf{n}}{\mathbf{l} \cdot \mathbf{n}}$$

If the line starts outside the plane and is parallel to the plane, there is no intersection. In this case, the above denominator will be zero and the numerator will be non-zero. If the line starts inside the plane and is parallel to the plane, the line intersects the plane everywhere. In this case, both the numerator and denominator above will be zero. In all other cases, the line intersects the plane once and d represents the intersection as the distance along the line from $\mathbf{l_0}$.

Figure 3. Line plane intersection - [http://en.wikipedia.org/wiki/Line-plane_intersection]

3.4.1. Damping and noise filtering

The tracking of the users head and hands might be noisy depending of the way the data is provided and processed. If the position of the head and the hand a both noisy a line through the two points will be even more noisy. A noisy jittering line will result in a not very stable point of intersection with a plane.

Running average and Kalman filtering

A simple method of smoothing is averagering, either using a windowed average or a running average filter. Another more advanced option is the Kalman filter. The Kalman filter build on a simple representation of a linear system where the output is equal to the input multiplied by a gain and added to the last output multiplied by another gain. A linear system like this is simple and known from for instance linear interpolation [17]. Eventhough the Kalman filter might give the best result, the simpler much running average might be sufficient for this purpose.

3.5. OpenNI

The following section contains information about the OpenNI functionallity likely to be useful for the project, and is based the OpenNI user guide [15].

The Nite middleware components currently supported with interest to this project are:

- Full body analysis middleware: a software component that processes sensory data and generates body related information (typically data structure that describes joints, orientation, center of mass, and so on).
- Hand point analysis middleware: a software component that processes sensory data and generates the location of a hand point
- Gesture detection middleware: a software component that identifies predefined gestures (for example, a waving hand) and alerts the application.
- Scene Analyzer middleware: a software component that analyzes the image of the scene in order to produce such information as:
  - The separation between the foreground of the scene (meaning, the figures) and the background
  - The coordinates of the floor plane
  - The individual identification of figures in the scene.

The full body analysis can be used for the line of sight interaction by tracking the head and hands of the user. The hand point capability can be used to implement a known well tested interaction method as a reference to the two proposed interaction methods. The scene is suited to let the users know when they are in the view of the camera and who is currently the active user.

3.5.1. Production Nodes

3.5.1.1. Production Node Types

Sensor Related Production Nodes

- Device: A node that represents a physical device (for example, a depth sensor, or an RGB camera). The main role of this node is to enable device configuration.
- Depth Generator: A node that generates a depth-map. This node should be implemented by any 3D sensor that wishes to be certified as OpenNI compliant.
- Image Generator: A node that generates colored image-maps. This node should be implemented by any color sensor that wishes to be certified as OpenNI compliant
- IR Generator: A node that generates IR image-maps. This node should be implemented by any IR sensor that wishes to be certified as OpenNI compliant.
- Audio Generator: A node that generates an audio stream. This node should be implemented by any audio device that wishes to be certified as OpenNI compliant.

Middleware Related Production Nodes

- Gestures Alert Generator: Generates callbacks to the application when specific gestures are identified.
- Scene Analyzer: Analyzes a scene, including the separation of the foreground from the background, identification of figures in the scene, and detection of the floor plane. that states whether it represents a figure, or it is part of the background.
- Hand Point Generator: Supports hand detection and tracking. This node generates callbacks that provide alerts when a hand point (meaning, a palm) is detected, and when a hand point currently being tracked, changes its location.
- User Generator: Generates a representation of a (full or partial) body in the 3D scene.

3.5.2. Production chains

In the Production Nodes section, an example was presented in which a user generator type of production node is created by the application. In order to produce body data, this production node uses a lower level depth generator, which reads raw data from a sensor. In the example below, the sequence of nodes (user generator => depth generator), is reliant on each other in order to produce the required body data, and is called a production chain. Different vendors (brand names) can supply their own implementations of the same type of production node.

3.5.3. Capabilities

Currently supported capabilities:

- Alternative View: Enables any type of map generator (depth, image, IR) to transform its data to appear as if the sensor is placed in another location (represented by another production node, usually another sensor).
- Cropping: Enables a map generator (depth, image, IR) to output a selected area of the frame as opposed to the entire frame. When cropping is enabled, the size of the generated map is reduced to fit a lower resolution (less pixels). For example, if the map generator is working in VGA resolution (640x480) and the application chooses to crop at 300x200, the next pixel row will begin after 300 pixels. Cropping can be very useful for performance boosting.
- Frame Sync: Enables two sensors producing frame data (for example, depth and image) to synchronize their frames so that they arrive at the same time.
- Mirror: Enables mirroring of the data produced by a generator. Mirroring is useful if the sensor is placed in front of the user, as the image captured by the sensor is mirrored, so the right hand appears as the left hand of the mirrored figure.
- Pose Detection: Enables a user generator to recognize when the user is posed in a specific position.
- Skeleton: Enables a user generator to output the skeletal data of the user. This data includes the location of the skeletal joints, the ability to track skeleton positions and the user calibration capabilities.
- User Position: Enables a Depth Generator to optimize the output depth map that is generated for a specific area of the scene.
- Error State: Enables a node to report that it is in "Error" status, meaning that on a practical level, the node may not function properly.
- Lock Aware: Enables a node to be locked outside the context boundary. For more information, see Sharing Devices between Applications and Locking Nodes.

3.5.4. Generating and Reading Data

Generating Data

Production nodes that also produce data are called Generators, as discussed previously. Once these are created, they do not immediately start generating data, to enable the application to set the required configuration. This ensures that once the object begins streaming data to the application, the data is generated according to the required configuration. Data Generators do not actually produce any data until specifically asked to do so. The xn::Generator::StartGenerating() function is used to begin generating. The application may also want to stop the data generation without destroying the node, in order to store the configuration, and can do this using the xn::Generator::StopGenerating function.

Reading Data

Data Generators constantly receive new data. However, the application may still be using older data (for example, the previous frame of the depth map). As a result of this, any generator should internally store new data, until explicitly requested to update to the newest available data. This means that Data Generators "hide" new data internally, until explicitly requested to expose the most updated data to the application, using the UpdateData request function. OpenNI enables the application to wait for new data to be available, and then update it using the xn::Generator::WaitAndUpdateData() function. In certain cases, the application holds more than one node, and wants all the nodes to be updated. OpenNI provides several functions to do this, according to the specifications of what should occur before the UpdateData occurs:

- xn::Context::WaitAnyUpdateAll(): Waits for any node to have new data. Once new data is available from any node, all nodes are updated.
- xn::Context::WaitOneUpdateAll(): Waits for a specific node to have new data. Once new data is available from this node, all nodes are updated. This is especially useful when several nodes are producing data, but only one determines the progress of the application.
- xn::Context::WaitNoneUpdateAll(): Does not wait for anything. All nodes are immediately updated.
- xn::Context::WaitAndUpdateAll(): Waits for all nodes to have new data available, and then updates them.

The above four functions exit after a timeout of two seconds. It is strongly advised that you use one of the functions, unless you only need to update a specific node. In addition to updating all the nodes, these functions have the following additional benefits:

- If nodes depend on each other, the function guarantees that the "needed" node (the lower-level node generating the data for another node) is updated before the "needing" node.
- When playing data from a recording, the function reads data from the recording until the condition is met.
- If a recorder exists, the function automatically records the data from all nodes added to this recorder.

3.5.5. Main Objects

3.5.5.1. Context Object

The context is the main object in OpenNI. A context is an object that holds the complete state of applications using OpenNI, including all the production chains used by the application. The same application can create more than one context, but the contexts cannot share information. For example, a middleware node cannot use a device node from another context. The context must be initialized once, prior to its initial use. At this point, all plugged-in modules are loaded and analyzed. To free the memory used by the context, the application should call the shutdown function.

3.5.5.2. Data Generators

Map Generator
The basic interface for all data generators that produce any type of map. Main functionalities:

- Output Mode property:
  - Controls the configuration by which to generate the map
- Cropping capability
- Alternative Viewpoint capability
- Frame Sync capability

Depth Generator
An object that generates a depth map. Main Functionalities:

- Get depth map:
  - Provides the depth map
- Get Device Max Depth:
  - The maximum distance available for this depth generator
- Field of View property:
  - Configures the values of the horizontal and vertical angles of the sensor
- User Position capability

Image Generator
A Map Generator that generates a color image map. Main Functionalities:

- Get Image Map:
  - Provides the color image map
- Pixel format property

IR Generator
A map generator that generates an IR map. Main Functionality:

- Get IR Map:
  - Provides the current IR map

Scene Analyzer
A map generator that gets raw sensory data and generates a map with labels that clarify the scene.

Main Functionalities:

- Get Label Map:
  - Provides a map in which each pixel has a meaningful label (i.e. figure 1, figure 2, background, and so on)
- Get Floor:
  - get the coordinates of the floor plane

[15]

Gesture Generator
An object that enables specific body or hand gesture tracking

Main Functionalities:

- Add/Remove Gesture:
  - Turn on/off a gesture. Once turned on, the generator will start looking for this gesture.
- Get Active Gestures:
  - Provides the names of the gestures that are currently active
- Register/Unregister Gesture callbacks
- Register/Unregister Gesture change

Hand Point Generator
An object that enables hand point tracking. Main Functionalities:

- Start/Stop Tracking:
  - Start/stop tracking a specific hand (according to its position)
- Register/Unregister Hand Callbacks:
  - The following actions will generate hand callbacks:
  - When a new hand is created
  - When an existing hand is in a new position
  - When an existing hand disappears

3.5.5.3. User Generator

An object that generates data relating to a figure in the scene. Main Functionalities:

- Get Number of Users:
  - Provides the number of users currently detected in the scene
- Get Users:
  - Provides the current users
- Get User CoM:
  - Returns the location of the center of mass of the user
- Get User Pixels:
  - Provides the pixels that represent the user. The output is a map of the pixels of the entire scene, where the pixels that represent the body are labeled User ID.
- Register/Unregister user callbacks:
  - The following actions will generate user callbacks:
    - *When a new user is identified*
    - *When an existing user disappears.*

3.5.6. Configuration Using XML

OpenNI uses an XML file for configuring the context. This file is loaded when the context is created and helps the system know what information should be available and which callback functions should be set up. The nodes needed is "Depth", "User", "Gesture" and "hands". Depth is needed since information about users in front of the camera is extrapolated from the depth image only, in this way the system does not rely on the rgb color image and is therefore less sensitive to changes

in ambient light and the surrounding in general. The user node is needed for this project since it is the intent to track the position of users and their individual body parts like "head", "right hand", "left-hand". The context configuration which does this can be seen in Code example 1.

```
<OpenNI>
  <Licenses>
    <License vendor="PrimeSense" key="0KOIk2JeIBYClPWVnMoRKn5cdY4="/>
  </Licenses>
  <Log writeToConsole="false" writeToFile="false">
    <!-- 0 - Verbose, 1 - Info, 2 - Warning, 3 - Error (default) -->
    <LogLevel value="3"/>
    <Masks>
      <Mask name="ALL" on="true"/>
    </Masks>
    <Dumps>
    </Dumps>
  </Log>
  <ProductionNodes>
    <Node type="Depth" name="Depth1">
      <Configuration>
        <Mirror on="true"/>
      </Configuration>
    </Node>
    <Node type="User"/>
    <Node type="Gesture"/>
    <Node type="Hands"/>
  </ProductionNodes>
</OpenNI>
```

Code example 1. OpenNi XML configuration

3.6. Conclusion of analysis

Kinect and unity seams like a solid combination for the implementation of the system. Unity offers tools for implementation of the graphical user interface, while allowing easy vector and matrix calculation for the interaction methods. Furthermore Unity comes in a free version which should support all the functionality needed. The Kinect camera is widely accessible and ships at only about 1.200 dkk. and with a fast growing online community it seams like the right choice for a project of this kind at the moment.

An interface for testing the proposed interaction methods which is also capable of using state of the art interaction methods for reference must be designed. A system for logging the results of the test must also be designed and implemented along with a test allowing users to use the different interaction methods in different configurations, to gain as much data as possible.

4. DESIGN

This chapter contains design of the test, interface, graphics along with a rough sketch for the software design.

For any design to turn out great it must have a purpose, a specific cause, a solution to a problem and address these in the simplest, cleverest and best possible way. Since the purpose of the interface is to test different interaction methods, the designed interface must present the user with an interface with different level of interface difficulty and the ability to switch interaction methods. While keeping everything else the same.

4.1. Interface design and graphics

To enable easy testing of the proposed methods a testing program must be designed. The interface of the program should be designed so the test persons will understand what to do merely by using the interface. This will make for better comparison of the collected data since all users have been given the exact same information.

4.1.1. The test interface

The interface should be designed with natural interfaces in mind, meaning that the test base of the interface should not be compared to navigating the interface with a mouse. Like the difference from a cellphone interface to a typical GPS user interface design. The Latter typically have larger buttons to enable easy access to the interface while driving a car. The same holds true here, the interface should be designed with large clear buttons to enable easy navigation of the interface. This is okay since the purpose of the test is not to test natural interfaces with the users body as a controller against a traditional mouse/keyboard navigated interface. The purpose of the test is the test how different natural interaction methods with the users body as a controller compare in different situations. To set the scene for the test interface in a believable and not unlikely environment, a media center style interface is choosen. The metaphors for the interface will be movie box covers and a hand visualazing the point of interaction.

Difficulty of the interface tasks

The tasks for the test users should be divided into different levels of difficulty which indicates different levels of demands for the interaction methods. The task for the users to performe will be to navigate the cursor to the right box cover button and hover the cursor over it for a fixed time.

- **Easy**: Small number of large buttons.
- **Medium**: Smaller buttons and larger number.
- **Hard**: Large number of relatively small buttons.

Examples of media center interfaces from the Microsoft Xbox can be seen in Figure 5. Figure 6. and Figure 7.

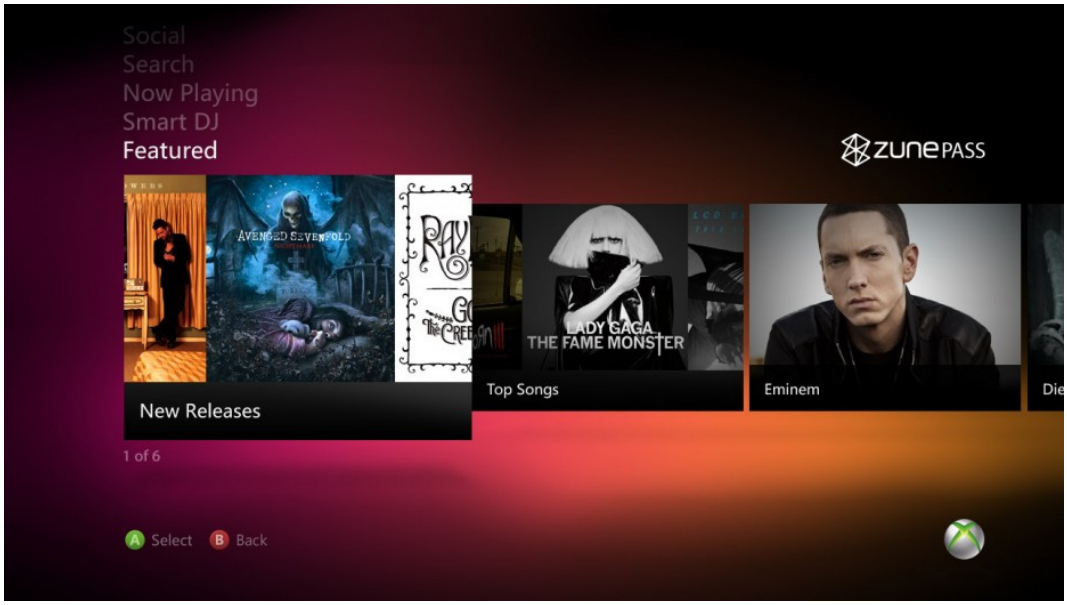


Figure 4. Xbox Zune user interface

Figure 5. Xbox kinect user interface

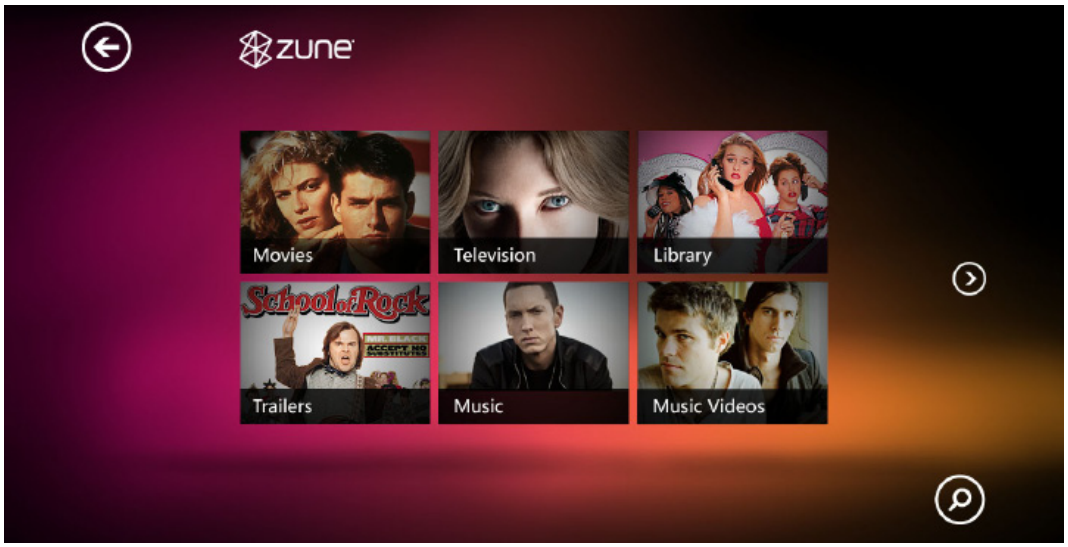


Figure 6. Xbox Zune user interface

An interface similar will be suitable for the design of the test interface. The user should select a movie and be presented with a new set of box covers, this task can then be repeated with different configurations of size and number of buttons. The test interface should be a believable situation, meaning that it should be a context where this type of interface is likely to be found. The interface should be somewhat familiar to the user, and present the test users with tasks familiar to the and task they are likely to preform in a real world situation. The edition of the test interface can be seen in Figure 8. To ensure that the users quickly learn to navigate the interface but does not memorize it, the position of the buttons are to change randomly. This is to ensure that the users are actually navigating the interface and not just following a previous learned pattern of interactions.

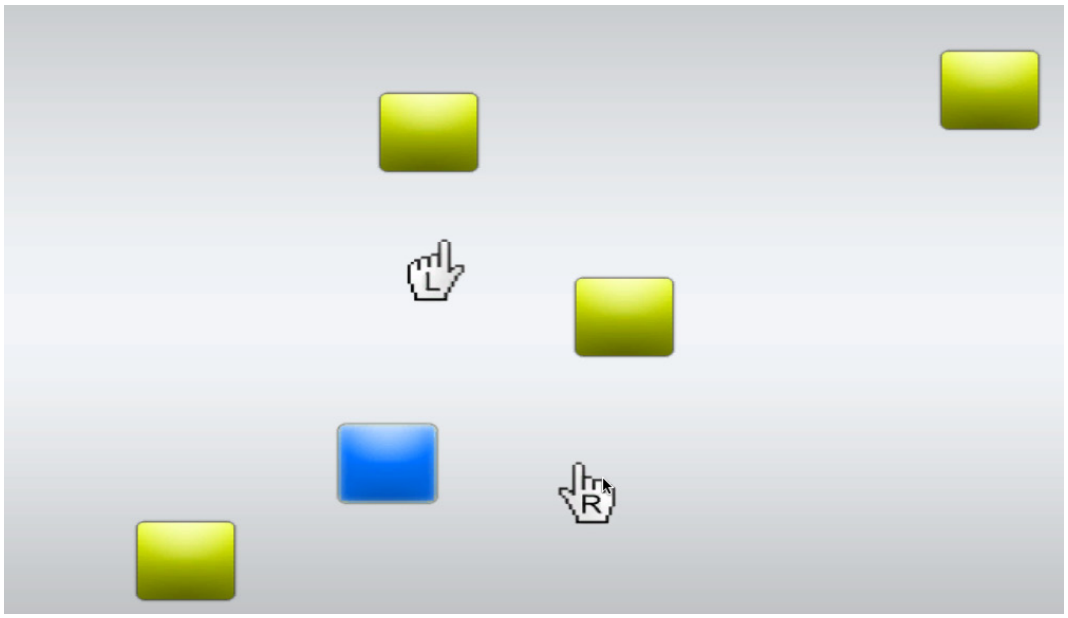


Figure 7. First implementation of the test interface

4.1.2. Buttons and interactive elements

The main interface for the test application contains an area in which a number of smaller button areas can be defined. The button areas inside the main area are arranged in a grid and are always present in a number to fill the grid. Between the individual buttons is a small gab to make the interface look better and ensure that all button are surrounded by non-interactive area. The main area containing the buttons must be of adjustable height and width to be able to tweak the implemented interface for looks and performance. The same hold true for the gab size which is also adjustable. These three variables along with the height and width resolution of the button grid dictates the resulting size of the buttons. The position of the buttons is relative to the main area and will therefore adjust accordingly. An example of the calculates button size and position can be seen below. The grid resolution is 2 by 2.

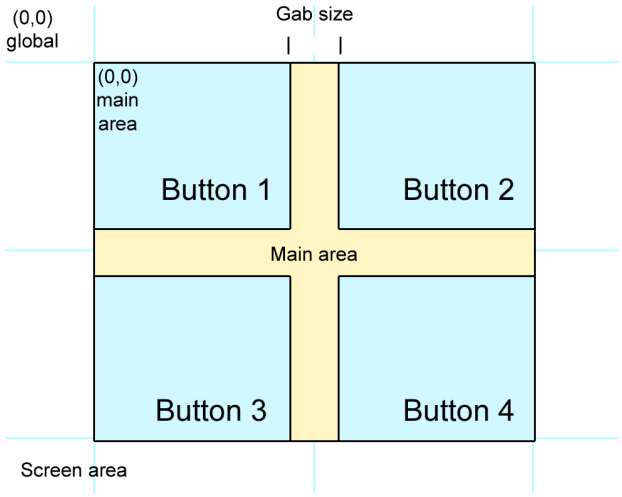


Figure 8. Wireframe interface mockup. 2 by 2 grid size. blue: button areas, yellow: main area.

In the example above the button width is equal to the main width divided by 2 and subtracted half of the gab width. In this case the height is the same calculation using main height instead of width.

4.1.2.1. Transitions, Animation and interpolation

To convey the message of a something being in progress on the interface, different visual effects should be used. This does not only make for a nice pleasing visual effect adding to the user experience, it does also provide the user with information. Alpha blending can be used to smoothly fade away GUI elements, so the users understands what is going on and GUI elements does not just disappear. Animation of the position and size of GUI elements should also be used to indicate that a the user has made an interaction and that something is currently happening or on the way to do so.

4.1.2.2. Indication of the interaction wait timer

To avoid another method of clicking and isolate the interaction of the moving the cursor, clicking on an object is done by hovering the cursor over an object for a specific period of time. Clicking and activating buttons like this is used the standard Xbox Kinect user interface. To indicate the time which has passed since the button area was activated and thereby the time left until the click interaction will take place, visual indication is needed. The initial idea was to change the cursor graphic. However

the movement of the cursor will confuse the message and not give clear indication of which gui-element is currently activated by the cursor. Instead the graphic state of the activated gui-element is changed. A hover-over state is added to each gui element to give immediate indication of the currently active element. A small time delay is added before the timer indication is activated. This is done to not stress out users when navigating the interface.

Figure 9. The three different states of the gui elements. Left: non-active icon, middle: activated icon, right: activated icon with interaction time indicator.

The wait for interaction time is a variable which can have large effect on the usability of the interface. If the time is to long, the user will be annoid and bored with the interface and it will seem stupid to hold ones hand still at on position for too long. On the other hand if the time is to short the number of unintented interaction will rise and thus ruin the usability of the interface. However since the purpose of this project is not to determine the perfect wait for interaction time, a suitable time will be found though internal investigation and test along with incorporating contemporary practice on the matter.

Interaction point and cursors

The point on the interface which is the actual point used for collision detection with GUI elements should be visualized for the user. This is typically done by well known mouse cursors, like an errow, a hand, an hour glas etc. In this case two things must be taken into account, the user must be able to clearly see and follow the cursor on the screen, while the cursor should convey the methaphor of being a human hand. The latter is important for the user to make the connection between the movement of the cursor and the hand in space.

Changing cursors to indicate different posibilities and ongoing processes might be a good idea, however for this interface which main purpose is to test interaction methods it might end up being just another factor of confusion for the users. The latter holds true since all the test persons are totally new to the interface and the whole experience of new interaction methods, eventhough the interface is designed to resemble a state of the art media center interface. The decision is made to provide users with additional visual feedback only on non moving elements, thus keeping the cursor the same at all times. Due to the different interaction methods and not least that they present an unfamiliar experience to the user, the cursor might be hard to control, and thus not a good place to supply important visual information. The cursor designed for the interface can be seen in Figure 11.

Figure 10. The cursor designed for the test interface

Visualization of user position information

If the users are to interact with the interface from different positions and angles to the interface, a way of letting the user know thier current position relative to the interface is needed. For this purpose a user radar is designed. To simplify later data processing the area visible to the camera is divided into fire discrete steps, which results in three different levels of user position difficulty.

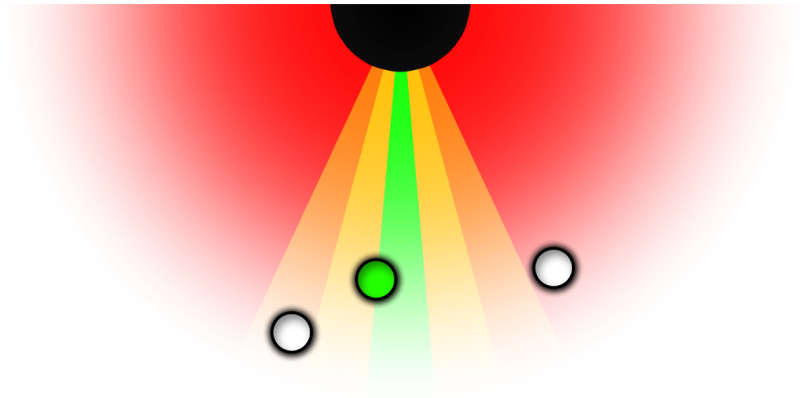


Figure 11. Graphical visualization of the found user position, with five different areas of angles to the interface.

As shown in Figure 12. the position of users relative to the screen plane which is the top edge of the figure. Inactive users are visualized by a white circle while the currently active user is visualized by a green circle. This distinction is made to show the user if they are currently tracked correctly during the test of the interface. The figure shows an example of a scene with three potential users and one currently active user. Only one user should be active at any time during the tests.

4.1.3. Graphics design

As mentioned in the introduction to the design chapter, the purpose of the graphical user interface is to blend into the context and seam believable to the test users. The graphical style and look of the interface is designed to be similar to state of the art media center interfaces. This style is chosen to set the test in a beliveable environment, and in a context where interaction like this is likely to be used in the future. The test is designed with different number and size of movie buttons, to test the interaction methods.

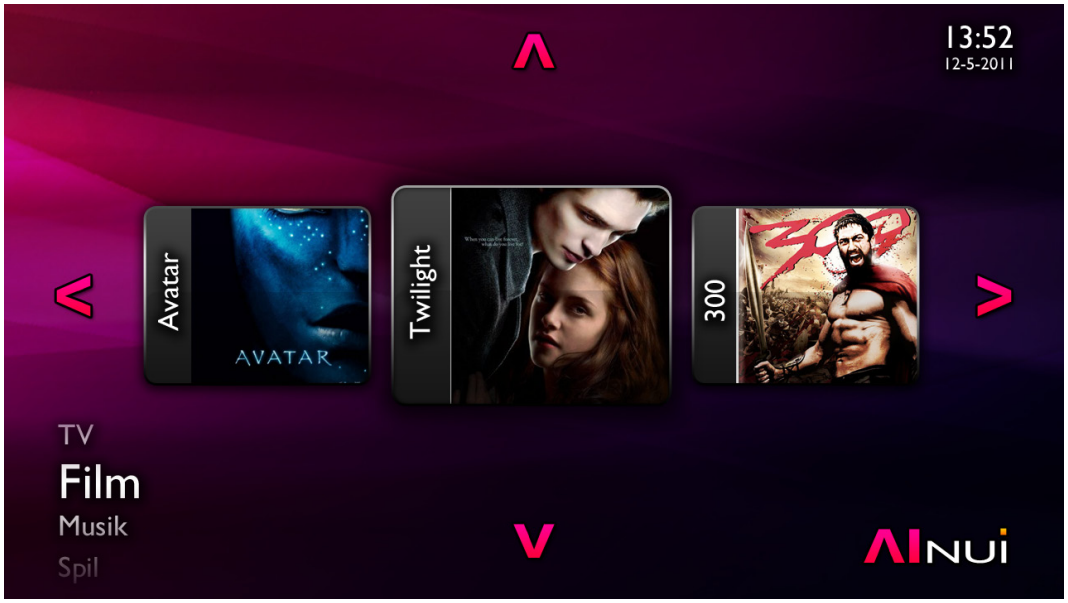


Figure 12. Simple one line movie icon interface design

First the interface was designed as more of a navigation task, enabling the user to hover over arrows near the four edges of the interface, resulting in the interface shifting in the chosen direction revealing new content, see Figure 13. Since the purpose of the interface is to test the interaction method, which is essentially the same whether a users wants to click an arrow or a folder like button pretenting to contain a certain movie, the interface was changed to only contain movie selection buttons.

The second iteration of the graphics for the interface can be seen in Figure 14. with a grid size 2 by 3 and in Figure 15. i a more difficult to navigate configuration with a total of 20 buttons in a 4 by 5 grid.

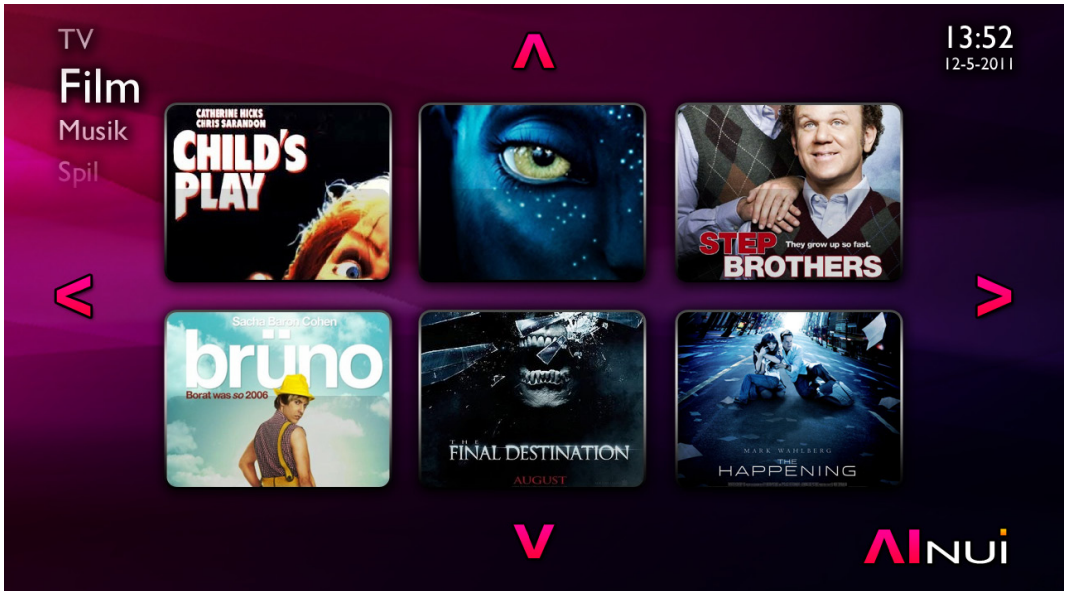


Figure 13. Large icons on the movie selector interface

Figure 14. Small icons on the movie selector interface

4.2. Interaction method design

The interaction method which needs to be designed are, "Line of sight" and "User area mapping" along with two already known interaction methods for reference and comparesons. The two known methods are conventional computer mouse or touch pad interaction, and the hand cursor interaction provided by OpenNI.

4.2.1. Mouse or track pad

The convensional way of controlling stationary computers and laptops today is by mouse, so this method is really already available to the system. The only change which is made is the substitution of the system cursor by the larger hand cursor designed for this interface. This is done to maintain persistance between the different tests where the user has to navigate the exact same interface but with a different interaction method. In this way the mouse test becomes a reference to the other tests, along with supplying a methods for familiarizing the users with the interface before the actual test of the proposed interaction methods.

4.2.2. Hand direct

The hand should be tracked as if tracked on a 2D image and directly mapped from a fixed area of movement in the 2D image to the area of the interface.

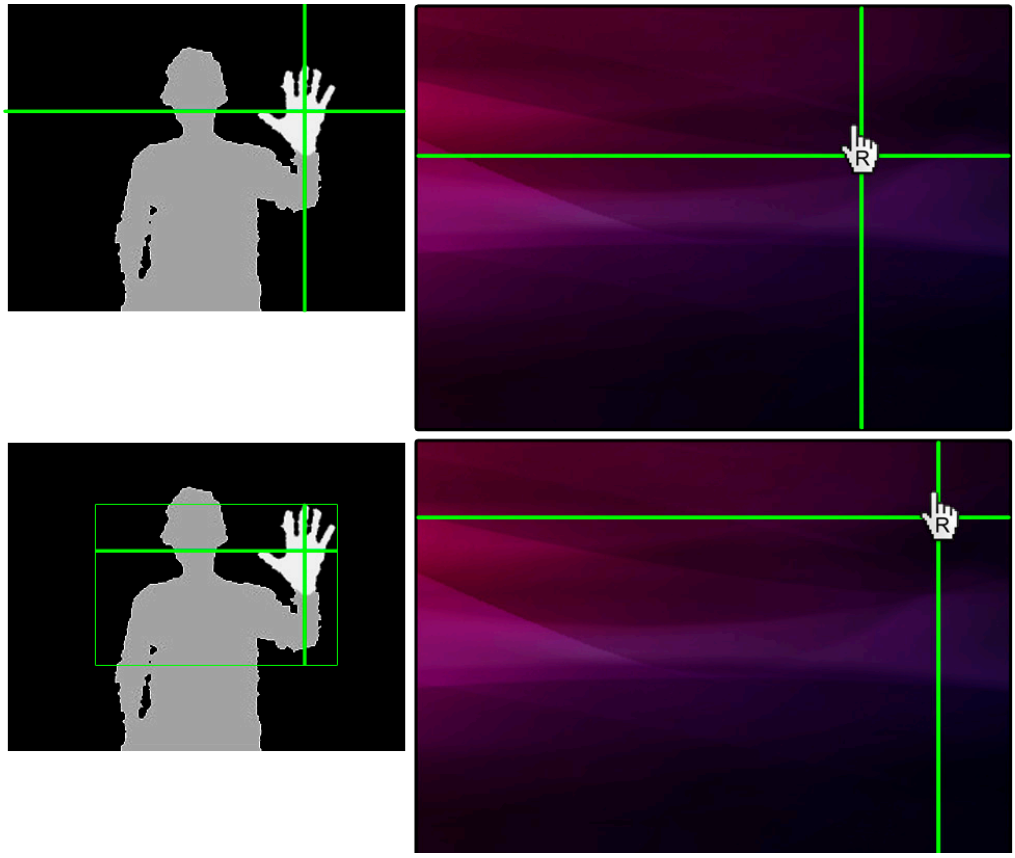


Figure 15. Direct hand position mapping concept

As shown in Figure 16. the position relative to image size is directly mapped from camera image to cursor position on the interface. This mapping could also be scaled to map a section of the camera image to the entire size of the interface screen.

4.2.3. Line of sight

The coordinated of the tracked user must be converted into real world coordinates, allowing direct usage of a screen with a known size in real world coordinates.The point af interaction is the point of intersection between the screen plane and a line through the head and hand position. Figure 17. shows the concept behind the interaction method.

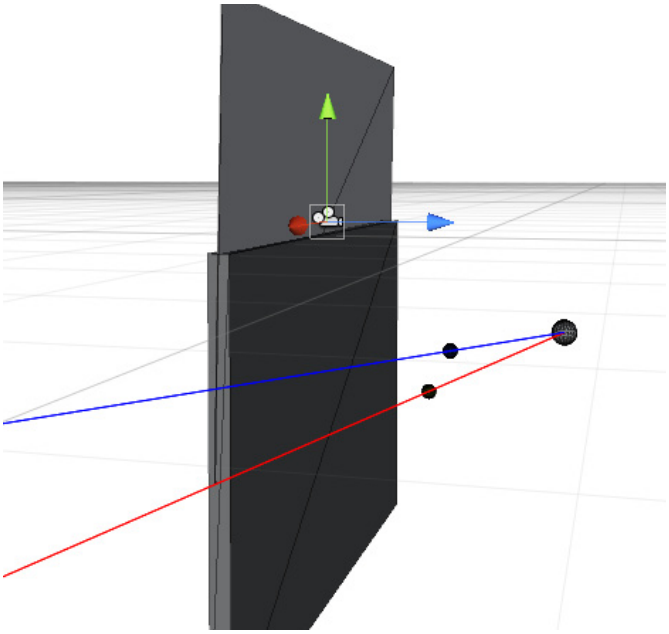


Figure 16. Concept setup of line of sight method

4.2.4. User reach area mapping

The problem with the direct mapping interaction method is that it is literally a direct mapping between the 2D image viewed by the camera and the 2D image shown by the screen. Even though the area can be scaled or offset it will always be somewhat fixed forcing the user to control the interface from a certain position in space. Since it is found possible to track not only the hands, but also the head and body of the user, this information can be used to transform the space needed by the interaction method to work to the area around the user in the area of reach directed towards the screen. A sketch of the concept seen from above can be found in Figure 18.

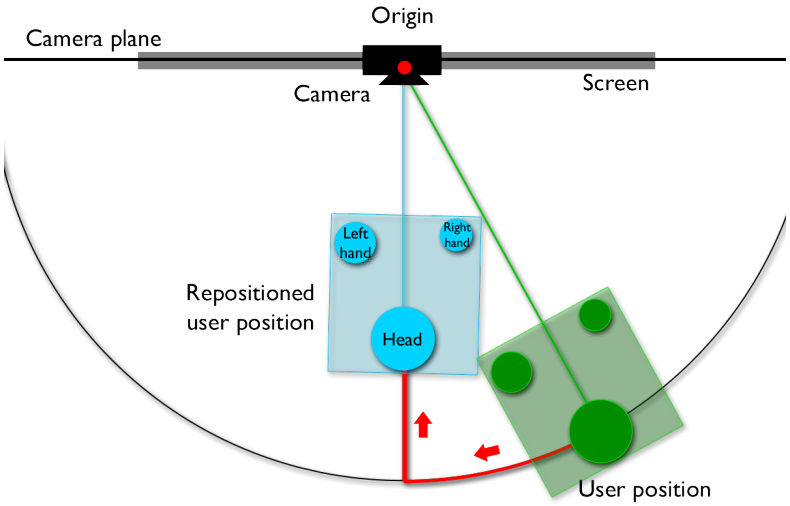


Figure 17. User reach area mapping interaction method concept diagram

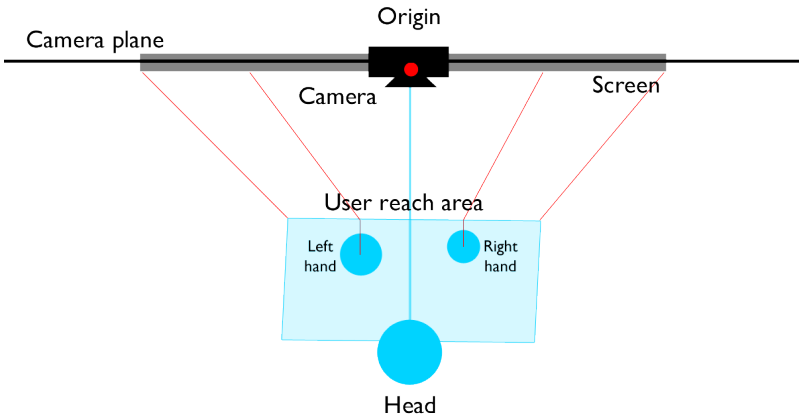


Figure 18. Mapping of user reach area to interface screen area

The user reach can be the actual reach area determined by the arm length of the user, a fixed area or a smaller ared scaled by the users arm length. See Figure 19.

4.3. Software design

The main flow of the program is as follows

- Initialization
- Show a randomly picked interface difficulty setup and randomize movie buttons
- Main loop (configurable setting are: interface difficulty, user position difficulty, interaction method)
  - If: cursor is over a movie button
    - *Start wait for interaction timer*
  - Else: stop active timers
  - If: timer finished
    - *Log results and re-initialize interface*

The interface difficulty will automatically randomize each time the interface is re-initialized, while user position difficulty and interaction method can be manually adjusted.

The movie button class
The movie button objects holds the name of the movie, along with the movie image, the size and position information. This design is chosen to enable the buttons to have changing size and position while still being easy keep track off. In this way whenever the area of the button is changed the graphics is automatically re-sized accordingly. When a button area is activated, the name and the graphics is easily associated with the interaction location.

4.4. Sound design

Auditory feedback for the test interface is designed to be a modest "on-roll-over" sound and a more distinct click sound. This will give the user information about that some one is interacting with the interface, even if the user did not intent to and might not even be looking at the screen. Two sounds must be implemented, a rollover and a click sound. The sounds must be distinct enough to catch the attention of the user will being as little annoying as possible.

4.5. Conclusion of design

The key point of the design which needs to be implemented is:
- A graphical interface capable of sizing and positioning buttons which should resembled movie box covers
- The interface must show an indication of the wait for interaction timer. This must not be done by changing the cursor but should be done on fixed GUI elements while still displaying information about which button are is currently active.

Test interface with graphical user interface.

- Adjustable interface difficulty, different size and number of buttons
- Indicate the task
- Indication of wait for interaction time

Graphics

- Buttons
- Cursor hover indicator graphics
- Wait for interaction timer indication graphics
- Hand cursors

Sound

- On roll over sound
- Interaction sound, distinguishable compared to rollover sound (like the click of a real button, metaphor)

5. IMPLEMENTATION

The following chapter contains detail about software and hardware implementation of the system. The Microsoft Kinect and the Unity game engine was chosen for the implementation. However the result should be reproducible, using other compliant depth sensing hardware and means of GUI visualization and vector calculation for the different interaction methods.

The test setup of the system is implemented using the following components:

- Microsoft Kinect sensor
  - Power adaptor
  - USB connector
- Phillips flat screen television
- MacBook Pro
- Unity (free edition)
- MonoDevelop
  - C# programming language
- OpenNI and NITE Middleware

5.1. Hardware setup

The system is set up with the camera located above the screen to ensure clear view of the users head and hands at all times. Were the camera to be placed below the screen a situation could arise where a hand would block the cameras view of the users head. The real world position and size of the screen is measured and noted along with the relative placement of the Kinect camera. This information is user by the system for the interaction.

5.2. Software implementation

The software implementation was as mentioned done using C# and the Unity game engine. The core of the system is responsible for the actual flow of the test application. Everything which is needed is initiated from here. The main functionality lies with in the main program flow which control the GUI interface, while the main collective data is gathered in a singleton called "ApplicationController". The singleton design pattern is used here to ensure that variables used several places can be updated and maintained only one place.

5.2.1. OpenNI and Nite

Everything related to the OpenNI context is implemented using a singleton design pattern and is therefore effectively only initialized once. When the instance is created a new initialization thread is started. This threading is done not to stall the main rendering thread in Unity which is essentially single threaded a far as what the application created using unity. When variables and function are tried access a static flag is tested to see whether the OpenNI context is created and thereby ready to use. When the context is created and another object wants to use an OpenNI context, an instance of the already created context will be returned instead of initializing a new instance. When the instance is created the "Init()" function is called from the constructor. What important to stress in Code example 2. is the file path to the "xml" configuration file. System paths in Microsoft Windows uses a backslash character "\" to denote a folder separator, while Mac OS X uses a forward slash "/".

```
private void Init ()
{
  // set path to xml setup file
  // if osx set path
  if (Application.platform == RuntimePlatform.OSXPlayer )
  {
    OpenNIXMLFilename = ".//" + OpenNIXMLFilename;
  }
  // if windows set path
  else if (Application.platform == RuntimePlatform.WindowsPlayer)
  {
    OpenNIXMLFilename = ".\\" + OpenNIXMLFilename;
  }
  // init thread setup
  initThread = new Thread (new ThreadStart (InitThread));
  initThread.Name = "Init thread ";
  initThread.Priority = System.Threading.ThreadPriority.Highest;
  initThread.Start ();
}
```

Code example 2. Initialization function called once when the context is created

It is importante to deconstruct and abort threads which has not finished when the object is destroyed.

```
// decinstructor
~OpenNIContext ()
{
  MonoBehaviour.print ("Destroying context");
  if (initThread.IsAlive) {
    initThread.Abort ();
  }
}
```

Code example 3. Deconstructor for the OpenNIContext class

The actual job carried out by the initThread is shown in Code example 4.

```
private void InitThread ()
{
  MonoBehaviour.print ("initThread started");
  MonoBehaviour.print ("Context creation started");
  this.context = new Context (OpenNIXMLFilename);

  // chech if context was created correctly
  if (context == null) {
    MonoBehaviour.print ("Context creation error!!!");
    return;
  }
  MonoBehaviour.print ("Context creation ended successfull");

  this.Depth = new DepthGenerator (this.context);
  MonoBehaviour.print("Depth image generator created");

  this.mirror = this.Depth.MirrorCapability;
  MonoBehaviour.print("OpenNI initiation done!");

  // Set flag to true,
 //letting other object know that a valid context is available
  validContext = true;
  Start();
```

```
}
```

Code example 4. InitThread, responsible for setting up the connection to OpenNI and the Nite Middleware

As seen in Code example 4. the last thing the thread does before finishing is to set flag to let other object know that a valid context has been created, and then call the "Start()" function, which can be seen along with the Update() function in Code example 5.

```
void Start ()
{
  if (validContext)
  {
    Debug.Log ("start valid");
    this.context.StartGeneratingAll ();
    ready = true;
  }
}

// Update is called once per frame
public void Update ()
{
  if (validContext)
  {
    //Debug.Log ("update valid");
    this.context.WaitNoneUpdateAll();
  }
}
```

Code example 5. OpenNIContext class, Start()- and Update()-function

The Start() function is by default called by Unity when the program is run. After the Start() function has been called the Update() function is called by the Unity engine once every frame. Another boolean flag called "ready" is introduced in the start function which is a public getter allowing other object with a reference to the class check of the context is ready in their own Update() functions. The Update() function on the OpenNIContext instance updates the created context if it is available. This WaitNoneUpdateAll() function updates the nodes setup in the "xml" configuration file, in this case the nodes seen in Code example 6.

```
<Node type="Depth" name="Depth1">
  <Configuration>
    <Mirror on="true"/>
  </Configuration>
</Node>
<Node type="User" />
<Node type="Gesture" />
<Node type="Hands" />
```

Code example 6. The OpenNI nodes used by the implementation

5.2.2. GUI implementation

GUI or graphical user interface in Unity is rendered as 2D images on top off and after the 3D scene. GUI is thereby a completely different system than the 3D part of unity, and different rules for implementation applies.

Scalable GUI and transformation

GUI elements are positioned using pixel coordinates, which of cause changes when the screen resolution changes. The GUI is there for designed for a predefine default screen size and transformed according to changes in screen size. The matrix seen in Code example 7. is a transformation matrix with a translation of 0 in all axis, a identity quarternion rotation, and a scale factor in the X and Y axis. The latter is found by dividing the actual screen size by a fixed set screen size. All GUI build to the fixed screen size will now be scaled to fix accordingly if the screen size is not the exact size the GUI was designed on. The default screen size for this project was chosen to be 1024 by 768 pixels.

```
GUI.matrix = Matrix4x4.TRS (Vector3.zero, Quaternion.identity, new Vector3 ((Screen.
width / defaultScreenWidth), (Screen.height / defaultScreenHeight), 1f));
```

Code example 7. C# - unity gui transformation matrix

When working with GUI in Unity it is important to know the origin of the coordinate system.

```
rightHandPosition = new Vector2 (Input.mousePosition.x / Screen.width * defaultScreenWidth,
(Screen.height-Input.mousePosition.y) / Screen.height * defaultScreenHeight);
```

Code example 8. C# - Unity gui coordinates

5.2.3. Positioning the test interface GUI

For the test as shown in the design chapter, three different difficulties must be implemented.

- 3 columns x 2 rows (easy interface)
- 4 columns x 3 rows (medium interface)
- 5 columns x 4 rows (hard interface)

The columns and rows are to be scaled to fit inside a common interface area. All coordinates a calculated from the top left corner of the interface area. In between the buttons there must be a gap which can be a fixed size or could be relative to the button area size.

```
// pseudo code for calculating the size of the button areas
Button height = (area height / number of rows) - ((gapSize
* number of rows - 1) / number of rows )
Button width = (area width / number of columns) - ((gapSize
* number of columns - 1) / number of columns )
```

Code example 9. C# - Button area size calculations pseudo code

The gap must be there top ensure that when the cursors exits one button area another button area is not immediately triggered. The position of the buttons within the area is relative to the top left corner of the area. The first button of cause is placed directly in (0,0) while the next in the first row will be placed in ((buttonWidth * 1 + gapSize), 0).

5.2.4. Calculation and animation of wait timer bar

The actual graphics is not scaled only the area in which the graphics is drawn, this results in a masking effect where only the part of the image covered by the gui box is shown. Now this masking box is gradually changed over time.

```
GUI.color = new Color (1f, 1f, 1f, 1f);
GUI.DrawTexture (item.rect, this.folderBgHover, ScaleMode.StretchToFill);

// begin gui group to act as a masking are
GUI.BeginGroup (new Rect (item.rect.x, item.rect.y, item.
rect.width * (1 - alpha), item.rect.height));

  // set the gui color, to ensure that the timer bar is non transparent
  GUI.color = new Color (1f, 1f, 1f, 1f);
  GUI.DrawTexture (new Rect (0f, 0f, item.rect.width, item.rect.height), timerGraphics);

GUI.EndGroup ();
```

Code example 10. Calculation of masking area effect for wait for interaction timer visualization

The alpha value is a number between 0 and 1 which represent the percentage of time which has passed since the cursor entered the button area. A small buffer time zone was added before the timer becomes visible and starts to grow from the left to the right across the button of the button. This was added to not confuse and stress out the user when unavoidably passing above buttons to get to the area of the desired button.

5.2.5. Consistency of the users experience

To make way for direct comparisons of the different interaction methods, every thing else about the interface stays the same. In fact every thing is exactly the same, on the interaction method is changed via a delegate. Again to ensure true compatibility between the different tests, only one parameter a the software will change for each test. This holds true for the interaction methods as for the difficulty of the user interface tasks which are also controlled by changing a delegate. As shown in the image above the buttons on the interface are placed randomly and only one of them are the one the test subject should click. When a user clicks any of the buttons regardless of color the buttons will be randomized again.

5.2.6. Sound implementation

Two sounds are created a rollover and a click sound. The sound files can be found on the appended CD and heard in the test application also found on the CD.

5.2.7. Implementation of interaction methods

Four different interaction method are implemented into the system, Traditional mouse interaction, hand point interaction, line of sight and another take at the hand point interaction called user area mapping.

5.2.7.1. Mouse or trackpad

To act as a point of reference while serving the purpose of familiarizing the test subjects with the test interface, a were simple and common mouse control implementation is used. To keep the overall look and feel of the interface a familiar as possible, the normal system cursor is substituted by the larger hand cursor used by the other interaction methods as well.

5.2.7.2. Hand direct

To enable direct comparison with state of the art interaction methods, the hand point direct mapping interaction method uses the same implementation used by the Xbox 360 supplied by the Nite middleware. The hand direct cursor control locks to the user when a gesture of moving the hand towards the camera is done. The interaction now takes place with this initial location as a reference. When a hand point is tracked a reference point is made. The cursor then moves with reference to this point until the user is lost or disconnected.

5.2.7.3. Line of sight

Since Unity is used for the implementation, the possibilities and special powers of a game engine is used to speed up and simplify the implementation. This means that the application can harness the power of the NVIDIA PhysX engine, which is an integrated part of Unity. PhysX enables easy access to collision detection between primitives such as, spheres, cubes, capsules, planes and lines. Since the line of sight method is designed to be the intersection between a planes representing the screen origin and a line through two points (head and hand), this can be achieved by using ray casting which can be done by a 3D point and a direction vector.

The calculation and mapping of the plane-line intersection point to a pixel coordinate on the screen, can be easy extrapolated using texture coordinates also supported the game engine. Rays is cast through the positions of the users hands. A screen rectangle is introduced into the screen If the camera is tilted according to the screen The screen rectangle in the screen is rotated correspondingly around the center point of the camera which is situated at the origin of the system in (0, 0, 0).

The rays cast through the hands are used to check for intersection with the screen rectangle. If there is an intersection the point of the intersection according to the size of the screen rectangle is calculated. The intersection point is converted into percentages. The percentages can the be applied to the actual pixel width and height of the physical screen used in the setup, whether it is a flat screen panel or a projector. The result is the pixel on the screen to which the user is pointing. Problems might occur when screen plane and camera plane are not aligned and parallel.

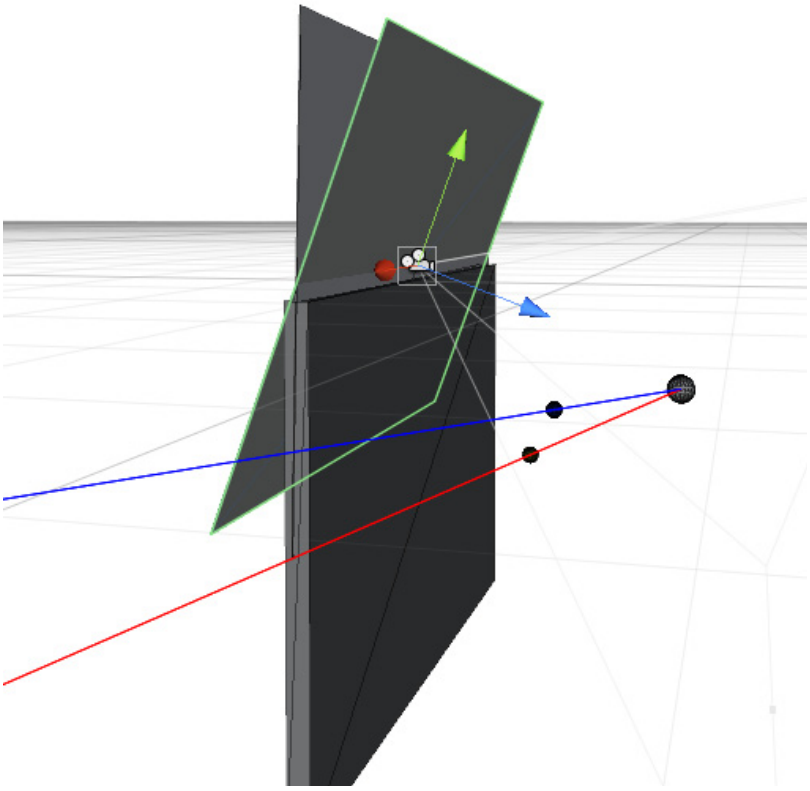


Figure 19. Concept setup of line of sight method, camera tilted relative to screen.

5.2.7.4. User reach area mapping

This method is implemented using the tracked point of the users body, "head", "right-hand" and "user-root". The user root is the general positing of the users body, and is the same position used to visualize the users position on the user radar. The positions of "head" and "right-hand" is now tracked relative to the "user-root" position. This means that even if the users real position is to the far left relative to the camera, the root can be translated and rotated as if the users body were directly in front of the screen. A control scheme like hand direct or even line of sight can now be introduced as if the user were standing directly in front of the camera (screen). An area of variable size is then define in front of the user relative to the users root position. When the users right hand is inside the area, the hand position is mapped to a 2D vector between 0 and 1 regarding to the known height and width of the interaction area in front of the user.

5.3. Conclusion of implementation

Unity was used for the implementation of the test software, which contains a test interface with buttons in the form of movie box covers. The graphical user interface is controlled using one of four different interaction methods, and everything about the test interface stays exactly the same when the interaction method is changed, meaning that essentially only the source of the cursors 2D position provider is changed.

The Microsoft Kinect camera was used as sensing device. The Kinect camera was chosen due to accessibility and the fact that the technology is now widely available to the masses. The Kinect works for the purpose but is to low resolution depth map to get reliable tracking of fingers, hand rotation etc. To safe time and focus at the task at hand, while enabling direct comparison OpenNI and Nite is used to track users and analyze the scene.

A believable graphical user interface is implemented, to set the scene for a realistic test environment.

6. TEST

This chapter contains the design of the test setup and procedure. The goal is to test the hypothesis stated in the introduction.

6.1. Test setup

The questions the test is designed to answer is if the method is fast and easy to use at different configurations of interface difficulty and relative user position.

The test users are first allowed to go through the assignments, using an interaction method they are already familiar with, like for instance a computer mouse, touch pad. This is done to familiarize the user with the interface and as much as possible take the interface out of the equation when comparing the performance and speed of the different interaction methods.

The test will be started out with a session of user exploration, to test whether the user initially finds the interaction intuitive. Since more than one method will be tested, all methods will be first impression tested with all users, however users will be presented with the different methods in a different order, which will be noted for later analysis. This is done to ensure that the first impression of one method does not influence the impression of the others. The accuracy and speed test will be done as one test, since the two are close related when it comes to the usability of the interaction and the interface. Hence a very fast interaction method with very low accuracy is just as bad as a very slow interaction method with very high accuracy. As mentioned about the first impression, interaction methods will be presented to different test persons in a different and random order. A test period will be given to the users to enable them to learn the interface, since this is not the issue of the test. Meaning that a users misunderstanding of the interface will effect the results of the interaction method.

6.1.1. Actual test

Pre-test - allowing the users to familiarize with the interface
First a test using a mouse where the user is presented with the different difficulty levels of the interface. Needless to say the position difficulty in not relevant in this part of the test.

The main test - the test of the interaction methods in different settings
The tests are presented with the different interaction methods in a random order, and are presented with all the different configuration of interface- and position difficulty for each interaction method. First a test of the interaction method where the user is not told about the method at all. This is done to investigate the users initial response to the interaction method. The the same interaction method, where the user is given information about the interaction method is now tested. The Interface give the user an assignment by showing the title of one of the film which box cover is displayed on the interface. The user must now find and click the movie button matching the movie title.

The tests can be redone with different settings
- Accept interaction time
- Button size
- Number of buttons

Data gathered from the tests

- The time for an interaction to take place
- Was the user successful in matching the right movie button to the movie title
- The position of the user in space relative to the interface

6.1.2. Test database and data gathering

A database for the gathered data is designed

Source	Supplied	Supplied	Supplied	Supplied	Test data	Test data	Test data	Supplied
Label	user nr.	Interface	Position	Interaction	Success	Time	Position	First test
Data type	int	int	int	int	bool	float	3D vector	bool

Table 1. Gathered data

The test id is an indicator of the actual test it self, and contain the following parameters

	Interface buttons:	Relative position:	Interaction method:
	Easy (0)	Easy (0)	"Mouse or trackpad" (0)
	Medium (1)	Medium (1)	"Direct hand point" (1)
	Hard (2)	Hard (2)	"line-of-sight" (2)
			"User near space mapping" (3)

The test id's can now be combined into codes like for instance "112", which means that the current test is on the easy interface, at the easy position and uses the "line-of-sight" interaction method. However for later data processing it might be preferred to separate the three test id variables into separate columns in the data base. Doing so will enable easy filtering on any one parameter without taking notice of the others. In this case the database design will be converted to the following:

Source	Supplied	Supplied	Supplied	Supplied	Test data	Test data	Test data	Supplied
Label	user nr	Interface	Position	Interaction	Success	Time	Position	First test
Data type	int	int	int	int	bool	float	3D vector	bool

Table 2. Database design 2. for test data gathering

Designing the database in this way has several advantages besides the ones already discussed. If a test subjects quits the test before completing the entire test with all the different configuration, the data gathered from the test person is still valid and usable, since the test is broken down to individual sub-tests in the task of clicking one button. This could potentially lead to a thinning and lack of date towards the end of the test, this however will not be the case since the order and the starting point of the test will be changed for each new test subject. The "first" flag is used to indicate if the current test is part of the test made during the first interaction method presented for the user. The mouse interaction is not counted as an interaction method in this sense, on the ones which are to be tested and compared.

The success variable

The success variable holds information about whether the user clicked the intended button on the interface. Even if the user by mistake clicks a button which is not the intended, the test is still logged and a new screen of buttons will appear on the screen.

The time variable

The time variable is used to log the time it took the user to find and click on a button since the last button interaction. As mentioned the test is logged even though the user did not succeed in clicking the intended button. The time spend on a non successful test can be used to determine if the user was actually looking to find the intended button, or by mistake held the cursor over a button for too long immediately after a new set of buttons has appeared.

The position variable

The actual position of the user is useful in test subjects were to stray from the intended position of the test, and will enable scatter plotting of for instance success rate for each interaction method according to relative user position.

6.2. Results

The entire data material gathered is available on the appended CD as a .txt file with 9 columns formatted as shown in Table 4. and one row for each movie button click test.

Source	Supplied	Supplied	Supplied	Supplied	Test data	Test data	Test data	Supplied
Label	user nr	Interface	Position	Interaction	Success	Time	Position	First test
Data type	int	int	int	int	bool	float	3D vector	bool

Table 3. Gathered test data format

6.3. Data processing analysis

Since the data is structured as a simple relational database the data can be sorted and segmented using one of the columns of supplied data of a combination of several columns. To keep a better overview of the data the data is labeled as follows:

Main labels

- User id
- Interface difficulty
- Position difficulty
- Interaction method
- Success
- Time (the time it took to find and click the movie icon)
- User x-position (relative to screen in real world coordinates)
- User y-position (relative to screen in real world coordinates)
- First test

Sub Labels

- Interface label
  - Easy (0)
  - Medium (1)
  - Hard (2)
- Position label
  - Easy (0)
  - Medium (1)
  - Hard (2)
- Interaction label
  - Mouse (0)
  - Hand direct (1)
  - Line of sight (2)
  - User area (3)
- Success label
  - Right (0)
  - Wrong (1)

The number found in parenthesis behind the label is the corresponding value at is represented in the database.

6.3.1. Initial data processing

Some initial data processing needs to be done to remove unusable samples from the database. Since data is only valid if the user clicked the movie buttons in a continues data series and as fast and accurate as possible. Since data is logged when a button is clicked, the data sample logged after a long break with no button interaction will show up as a relatively long time sample in the data.

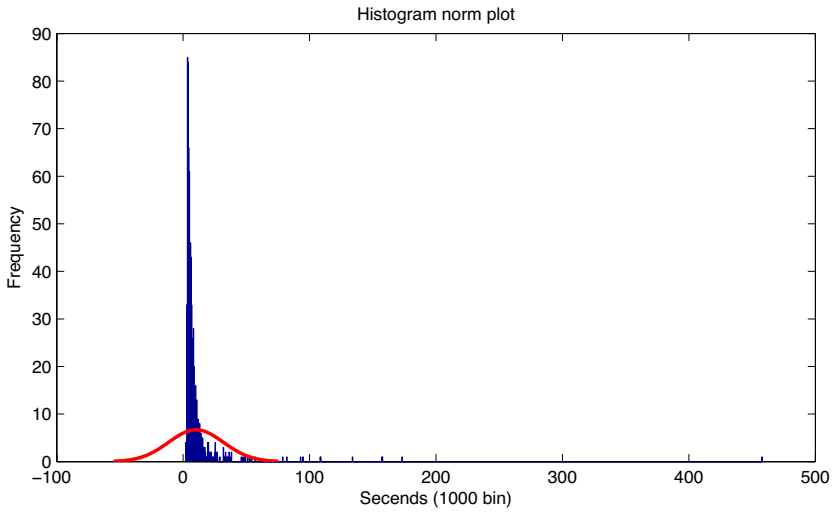


Figure 20. Raw data - time sample histogram - 1000 bins

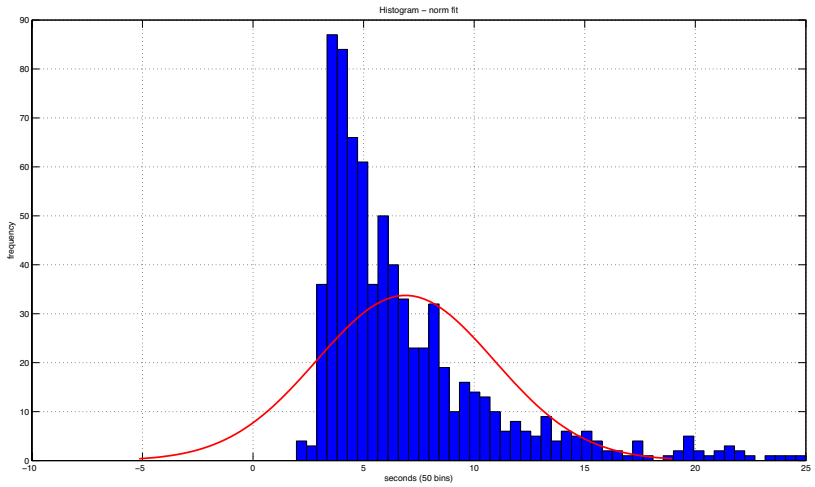


Figure 21. Histogram - time data - all samples - threshold 25 second - 50 bins

6.3.2. Test material

The tests were conducted partly as a quantitative and qualitative test in the sense that the users were asked to think out loud during the test. The quantitative part takes place during the test and is carried out automatically by the program. The qualitative evaluation takes place during the test and a free discussion following the test. The tests is done as expert tests, and all test user have some knowledge of the under lying technology. The qualitative data will be discussed in relation to the analysis of the quantitative data at the end of the chapter. In the case of the quantitative data, the variance of the average results for each user is compared to see to what degree use of further unique test persons would result in redundant data.

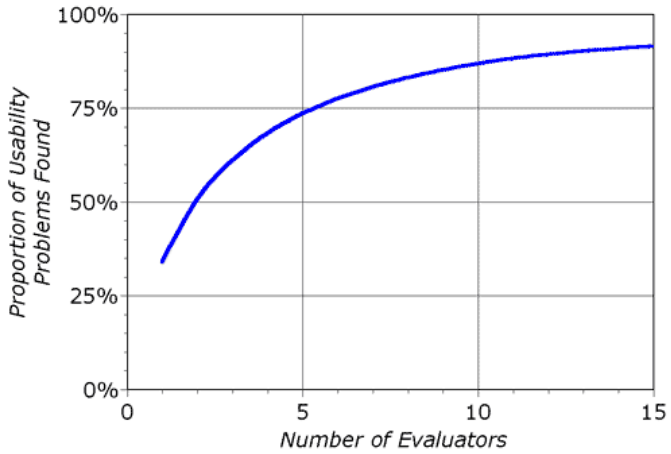


Figure 22. Relation between evaluators and information gain [http://www.useit.com/papers/heuristic/heuristic_evaluation.html]

The principle of ration between new problems according to number of evaluators, can be applied to the amount of new information according to number of expert users.

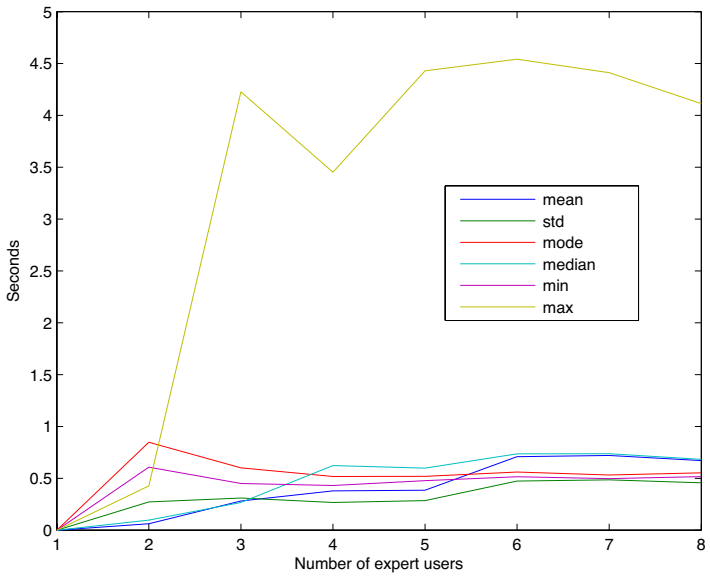


Figure 23. Standard deviation of N number of test users (Time measurement)

As Figure 24. states standard deviation tends to stagnate with the number of expert users rising. This means that as the number of users increase the amount of information gain decreases.

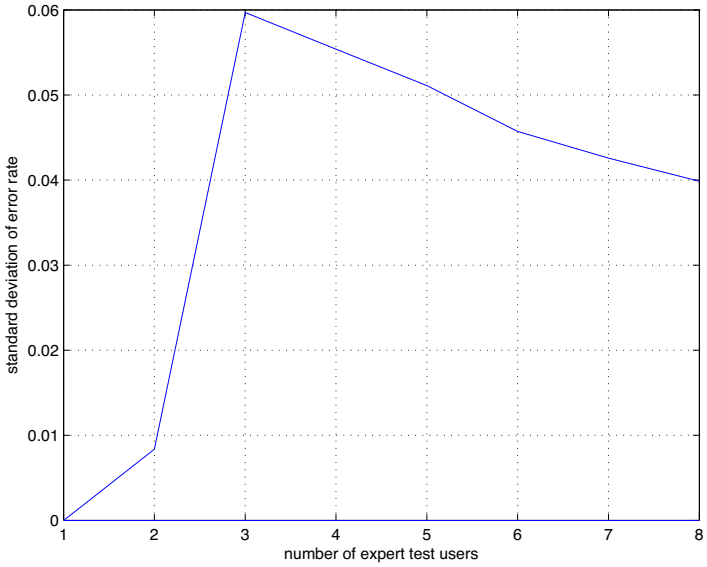


Figure 24. Standard deviation of error rate according to number of expert test users

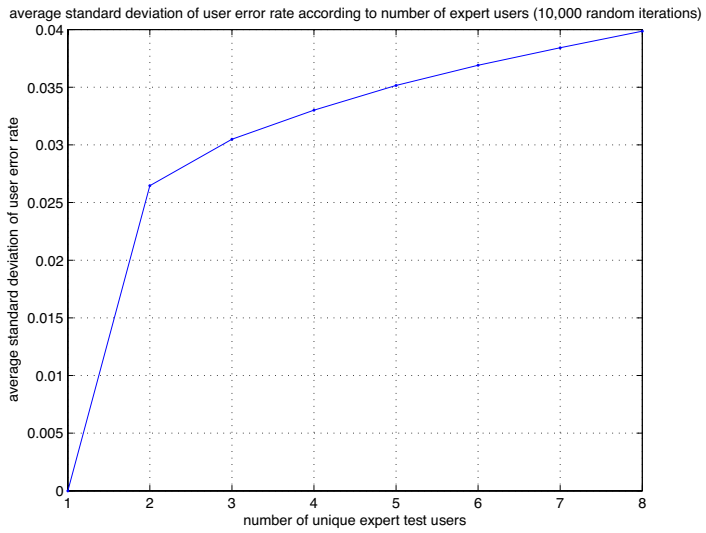


Figure 25. Average standard deviation of user error rate according to number of expert users (10,000 random iterations)

6.3.3. Classifying data

Since the data is structured the way it is, analyzing the data with respect to any of the other supplied parameters is trivial. The column of respectively user id, interface difficulty, position difficulty, and success, can be directly used as class labels since they are all of the integer data type.

The data is analyzed using Matlab and the PRTools toolkit [21], hence terms known from the field of pattern recognition like "class", "feature" and "dataset are used. The term "feature" are used as a description of the different columns of the dataset, while "class" is used to denote a separation of the dataset by the values of one of the columns. An example of this could be classification of the entire dataset by the "success" variable, this will result in a dataset with two classes respectively labeled "Right" and "Wrong". The information about user x- and y-position can then be used as features and plots like shown in Figure 27.

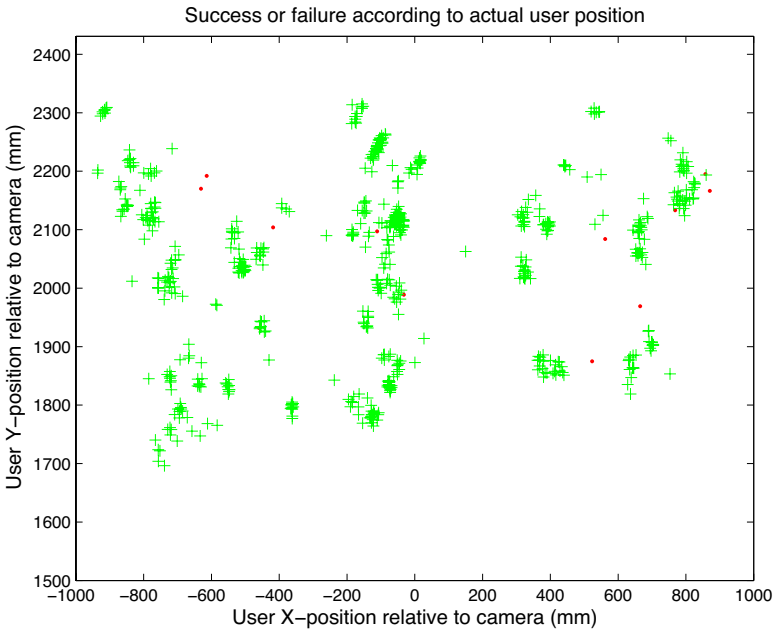


Figure 26. Example of plotting data using PRTools - Success according to x- and y-position

To be able to compare the different interaction methods the entire dataset is classified by the interaction method variable and then divided into sub-datasets according to the classes, meaning that each interaction method now has its own dataset only containing data gathered using this interaction method.

Error rate

The error rate for each interaction method is calculated by filtering the data on the class label "interaction label". This results in four new data sets all with the following structure, but all of the same interaction method:

The success data from the datasets now only containing a single interaction method, can be easily turned into an error rate in percentage by dividing the number of "false" success by the total number of rows in the dataset. With the datasets now divided into separate datasets for each interaction method, another interesting fact is to see the error rate according to the position difficulty and the interface difficulty. To do so the error rate is calculates as stated above for each of the 9 combination of position and interface difficulty.

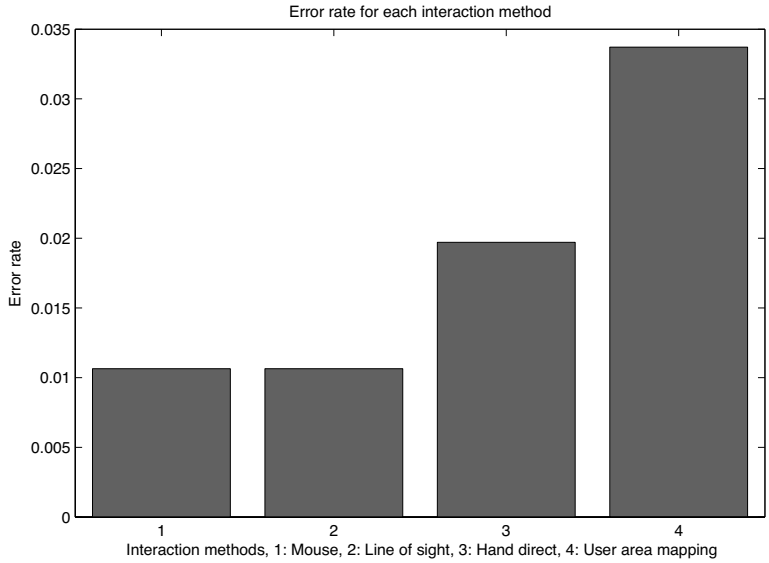


Figure 27. Error rate for each interaction method, all test users included

Position data

The actual real-world position of the user is logged when a click has happened, regardless of whether the user was asked to use the easy, medium or hard position relative to the setup. The interesting thing here is if the test user had to move away from the desired position in order to successfully navigate the interface. The data can be now be grouped into classes using the information about position difficulty and visualized with a color for each difficulty level on a scatter plot of the users actual real world position seen from above.

The raw position data gathered by the soft is formatted as a float between 0 and 1. A "X" position of 0.5 is directly in front of the camera while 0 and 1 are respectively 3 meters to the left and the right. A "Y" position of 0 is where the camera plane is and 1 is 6 meters away from the camera plane. Figure 29. Shows a scatter plot of real world x- and y-position relative to the camera from the entire dataset (test data 1.). As the figure indicates, the data I very reliable and does not need any further filtering. This is due to the fact that the position of the user is only logged if the system has a reliable tracked user at the time of the click interaction. If a click happens without a calibrated user tracked, the position of the user is logged as (0,0) and can therefore easily be ignored for data analysis and visualization.

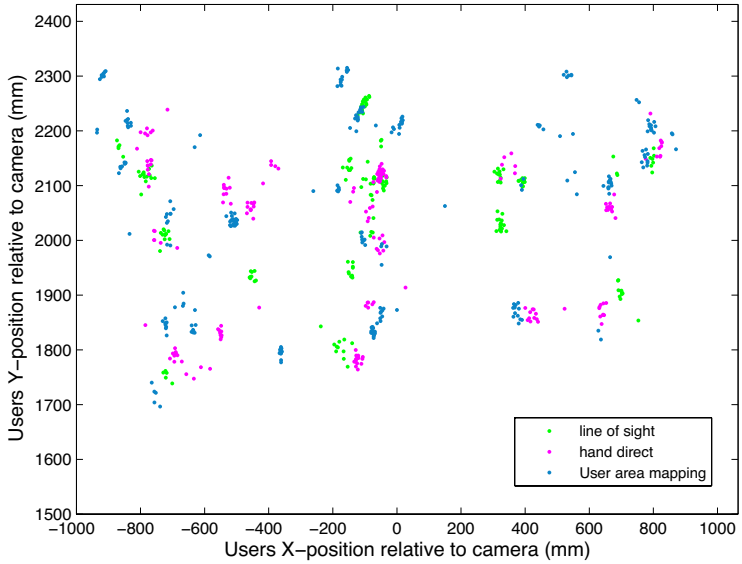


Figure 28. User position with respect to class (interaction method)

It must be remembered that position data for the mouse interaction class is irrelevant.

Time

With the error-rate for each interaction method now covered it is time to see whether a link between speed and error rate exist. Again it is interesting to compare the different interaction methods to see which was on average the fastest.

The minimum, maximum, average, mode, median and standard deviation is calculated for the time data for each interaction method. However before this is done outliers must be removed to get a more reliable results of the actual time it took the test users to find and click the right movie button after a new random set of buttons is presented. This is necessary since the software logs the time since the last click at all times, meaning that the moments the system is initiated the timer automatically starts. When a user pauses the test to talk or the test is in any other way disrupted, the timer will continue running and thus result in the following click to log a relatively long time. In other words the time data i only valid if the a users clicks a succession of buttons as fast a possible.

The histogram shown in Figure 21. indicates that the time data clusters in between 0 to about 30 seconds, for the user to find and click the next movie button. This means that data above a threshold can be viewed as irrelevant and removed. The threshold value can be a fixed time chosen to be a reasonable time for at user to have completed the task, or can be dependent on the standard deviation of the time data.

Statistics

The statistic data are plotted together in bar charts for each interaction method

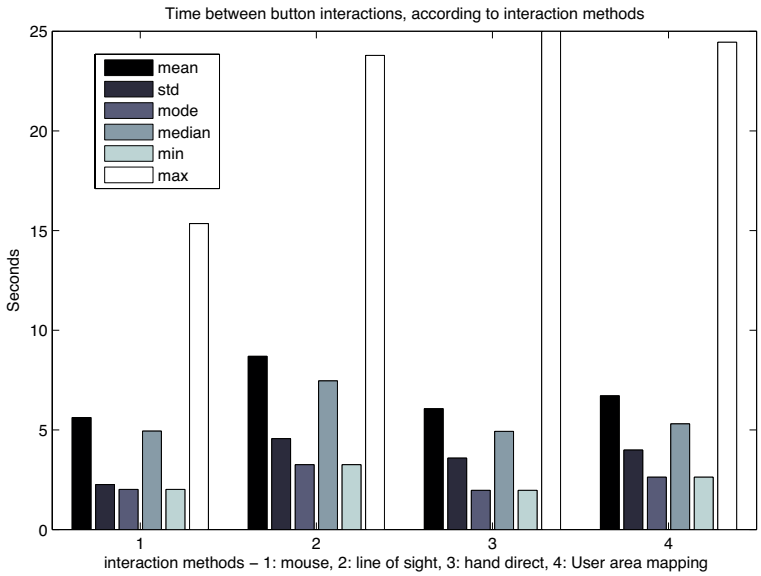


Figure 29. Time statistics for each interaction method

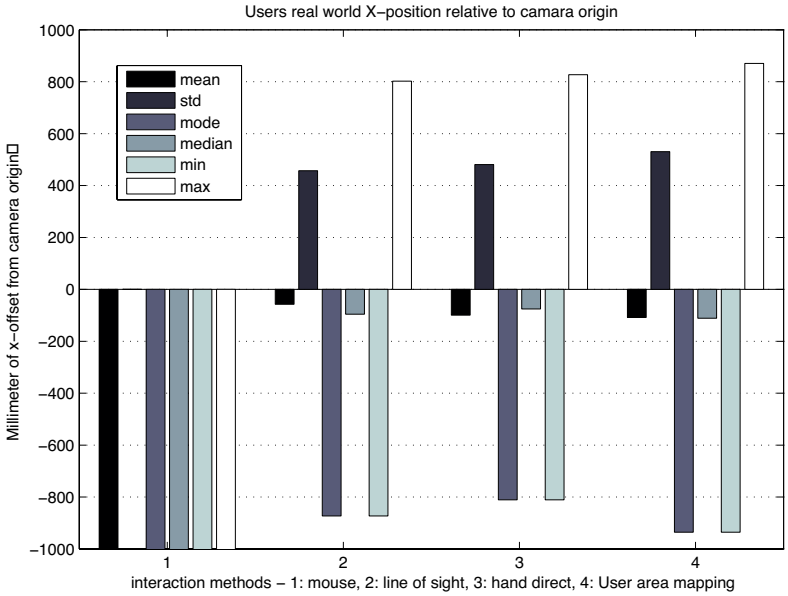


Figure 30. X-position statistics for each interaction method

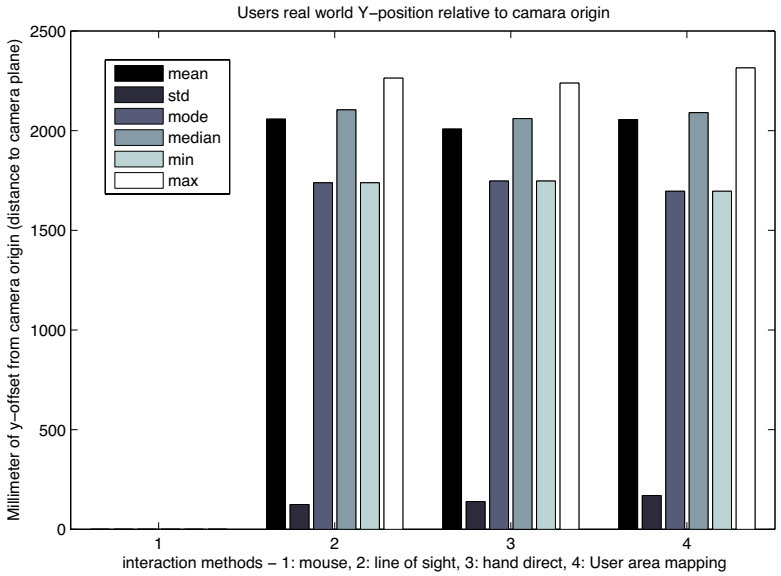


Figure 31. Y-position statistics for each interaction method

The data regarding time and the x-position are the most important, since the y-position data from the test is very stable at around 2 meters away from the camera.

6.3.3.1. Classifying data after position difficulty

To test if users are driven to stray from the initial position they are given at the start of the test, the dataset is classified according to position difficulty. The data can now be plotter as a scatter plot of user real world x- and y-position color- and shape-coded by the position difficulty.

```
% pick up class-labels (position difficulty)
LABS = +prdataPosition(:,3);
% add class labels to dataset
prdataPosition = dataset(prdataPosition,LABS);
% set class labels type: interaction methods
prdataPosition = setlablist(prdataPosition, positionLabel);
% pick user x-position and y-position for scatterplot visualization
[g,j] = seldat(prdataPosition,[], [7:8]);
figure();
% Create xlabel
xlabel('User x position relative to camera');
ylabel('User y position relative to camera');
% Create title
title('Position difficulty according to actual user position');
% draw scatter with respect for position difficulty class
scatterd(g);
```

Code example 11. Matlab - classifying data according to position difficulty

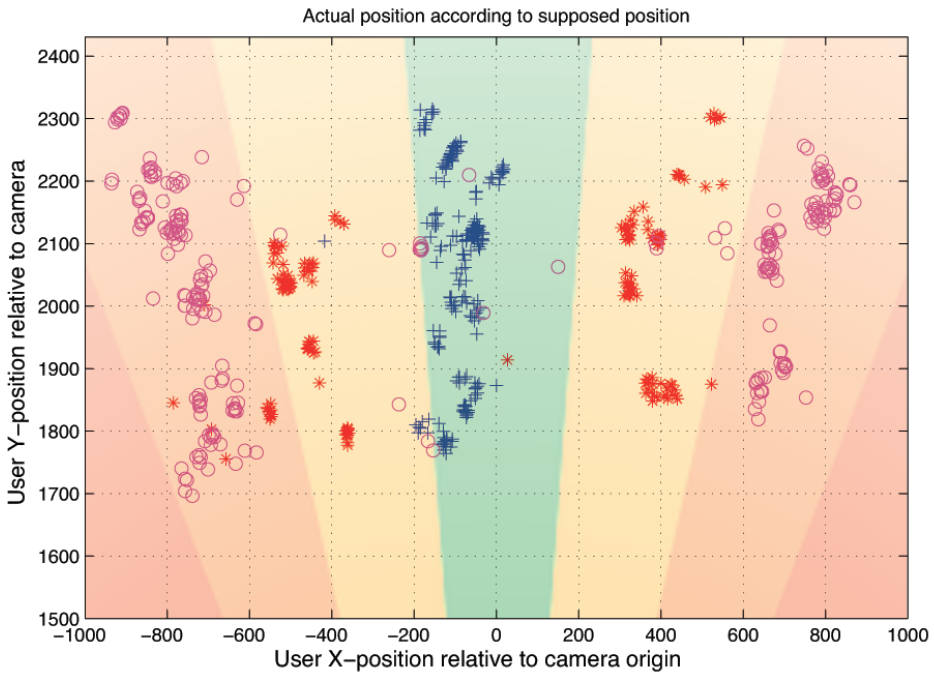


Figure 32. 2D real world user position scatter plot with color indicated position difficulty

As shown the data clusters nicely around the areas of position difficulty the users were asked to use. Users did not stray much from their supposed positions.

6.3.3.2. Comparing average time and error rate

To gain easy overview of the 36 different configuration, and their resulting average time and error rate. The average time and error rate for each of the 36 possible combination are isolated and plotted to gain easier access to a direct comparisons. Since performance of the different interaction methods first of all is a balancing between speed and accuracy. However this is not the entire truth, performance at different angles and interface difficulties must also be examined.

6.3.3.3. Investigating performance in all system configurations

A compromise between speed and accuracy is assumed to be the optimal solution, with this in mind the average time for all combination for each interaction method is visualized using a horizontal bar chart. The combined performance can be investigated by comparing the average time scaled according to the error rate. The scale can be done in different ways, depending on how errors are to be punished. The combined performance estimation found usable for this project uses a division of the average time by "1" subtracted by the error rate. The result of this calculation is that if the test has an error rate of "0" the combined performance will be the average time measurement, however if the error rate closes in on 100% the indicator of the combined performances will rise towards infinity. The of cause means that the lower the combined performance indicator is the better.

```
% punish methods on the time parameter by scaling using the error rate
MeanTime ./ ( 1 - ErrorRate )
```

Code example 12. Matlab - scaling time according to error rate

The values for measured average time, avarage minus standart deviation, error rate and the calculated combined performance is shown as respectively blue, purple, red and green in Figure 34., Figure 35., Figure 36. And Figure 37. The four figures each illustrates the performance of one interaction method in all possible configuration of position- and interface-difficulty.

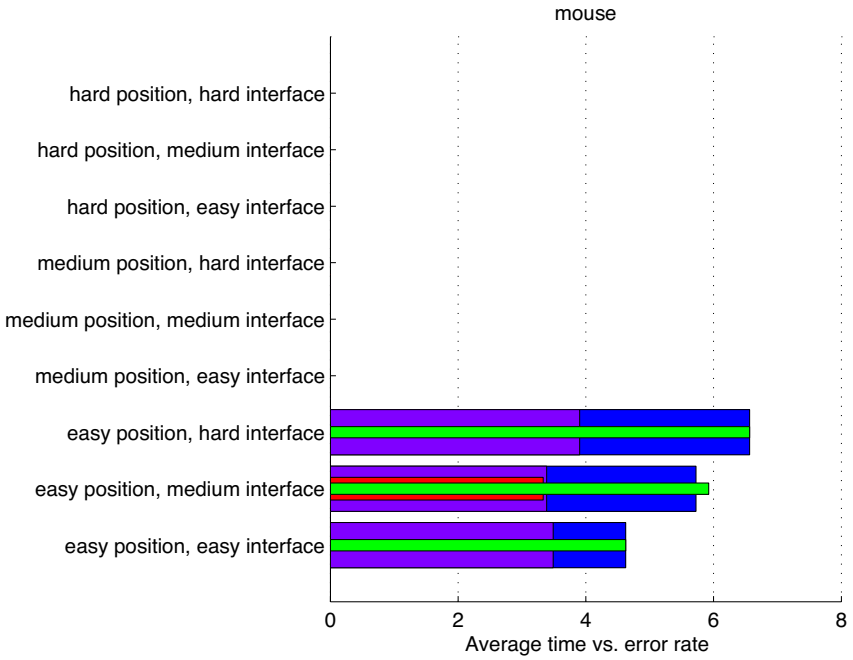


Figure 33. Bar chart - Error rate for all combination of position and interface difficulty (mouse)

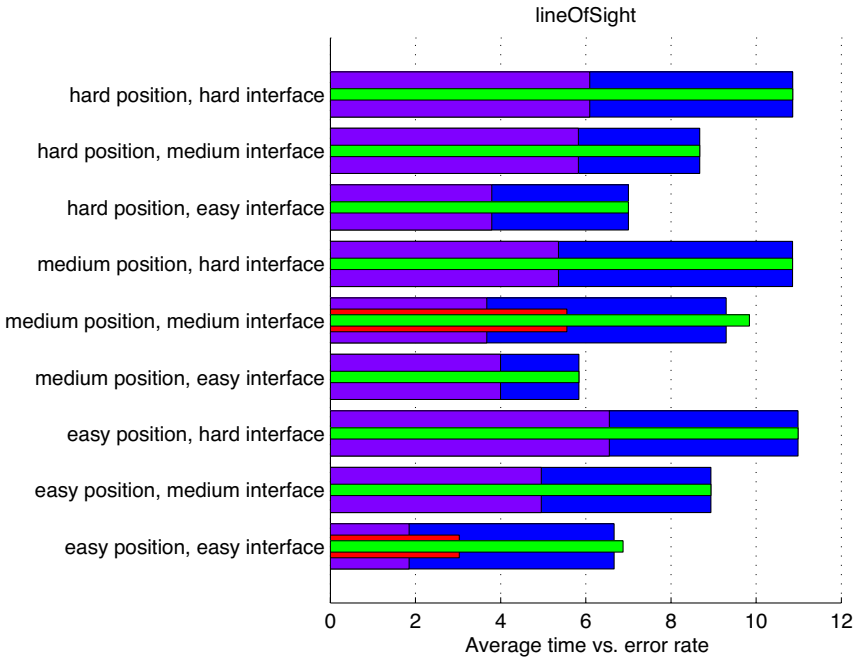


Figure 34. Bar chart - Error rate for all combination of position and interface difficulty (line of sight)

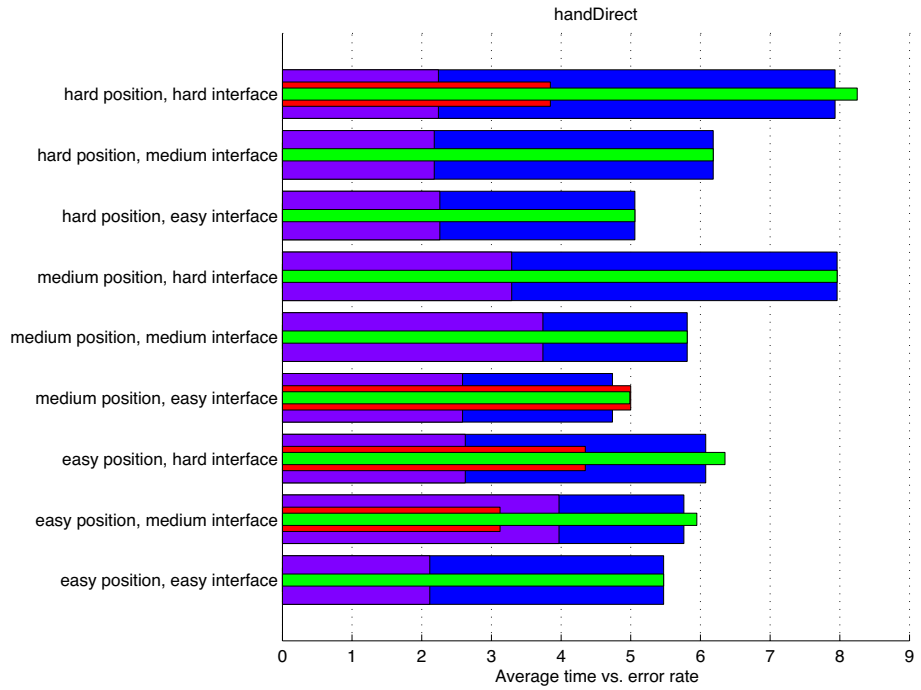


Figure 35. Bar chart - Error rate for all combination of position and interface difficulty (hand direct)

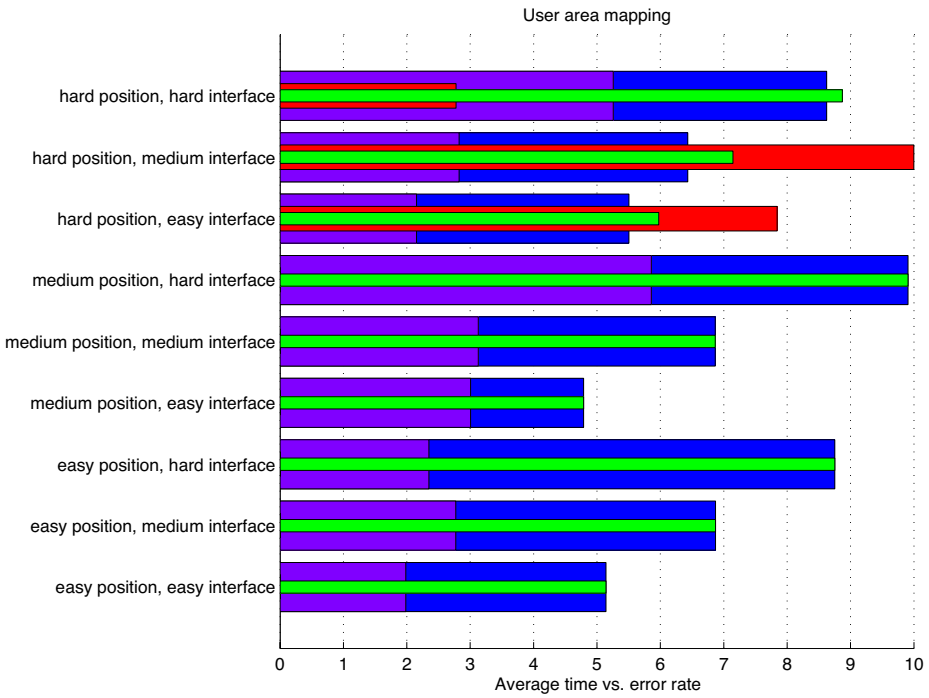


Figure 36. Bar chart - Error rate for all combination of position and interface difficulty (user area mapping)

6.3.4. Qualitative results

During the test the users were asked to think out loud and general speak their mind about the experience.

Discussed during testing

Line of sight

- Must have better and faster smoothing
- Hard to control with the hard interface
- Too much smoothing, seams unnatural, else to noisy
- Blocking cursor with hand
- Better when explained
- Hard on the arms
- Tends to keep arm stretch
- nice to ensure that the cursor does not click something by accident
- Test persons felt tired in their arms, using line of sight
- Some users found it tiring for the arms

Hand direct

- Initially better control than line of sight
- More direct feedback, hand eye coordination.
- Less noisy
- Some times getting court to an unwanted reference point, when moving root position
- OK also with the hard interface
- Less tired using hand direct interaction, bend arm

User area mapping

- Initially hard to understand - much better once the technique behind is explained
- More pleasant for the arms.
- Can interact with arm bend
- More pleasant interaction area in front of body
- Expected to work a little different at the hard positions (angle)
- In general problems when users were to far to the left relative to the camera.
- Too small interaction area in front of user.
- What is to happen when the users starts with the hand outside the active area... Indication of where the cursor is even if outside the screen area.

Discussed after test session

- Was the wait for interaction time suitable?
  - Small buffer time before wait for interaction timer visualizes
- Sound? Suitable? noticeable? annoying?
- What happens when a new interface presents it self and the user happen to have the cursor on one of the new active areas?

The users found the time fitting, and did not notice the pre wait for interaction buffer time, but did not report it missing, and were glad that it was there once they were told about it.

Mentioned by the users

Good that the interface starts to made objects not selected during wait for interaction, could even desaturate colors of other GUI elements.

Good and clear click sound

Even larger and clearer cursor

7. EVALUATION

Evaluation, conclusion and reflection on possibilities and future work.

The performance of the mouse interaction is not be analyzed since this test is only conducted to familiarize the users with the interface. However the average time and error rate for mouse interaction can be used as a reference and mean of comparisons, but not for the individual configurations of position difficulty.

Overall the easy interface performs best, this however is to be expected since the interface only presents the user with six different possibilities and therefore also shortens the cognitive process of locating the right button and not only the physical interaction of locating the right button. However it is clear from the data that the line of sight method performed equal regardless of position difficulty. The overall relatively poor performance of the method is likely to be due to noise as a result of the quality and resolution of the depth information and amplified by the fact that both noise from the hand and head position are noisy.

Thought the number of expert test users might be efficient for correcting interface- and interaction technical shortcomings, a better estimate of the error rate would have been better with a larger and wider user population. The wait for interaction time, though found to be suitable by the users might have been to high for this sort of test, meaning that navigating the interface with success is simply to easy. This results in very few errors because the users were able to correct in time, however this error should show up in the data as a increase in average interaction time.

As far as answering the question of the hypothesis, the data does not directly supply clear indication that the new methods provides natural and intuitive interaction from all angles. Though the interface was found believable and easy to overlook, the investigations shows that given the current setup it is not possible to totally omit visual indication by cursor on the screen.

In retrospect the decided wait for interaction time might have been too long for the purpose of testing interaction methods. Having a shorter time would have had the effect of provoking more errors, and thus revealing pro and cons of the different interaction methods.

The kinect camera turned out to have a surprisingly narrow field of view depth image, which has limited the magnitude of the result diversity. The fact that the Nite implementation of hand cursor "Hand direct"-interaction, has the potential to both fail and succeeded at all angles depending on whether the hand stays tracked when the user moves from one position to the other.

8. REFERENCES

1. Ros.org
   http://www.ros.org/wiki/kinect_node

2. Flow-the-psychology-of-optimal-experience
   Source: http://www.econsultant.com/book-reviews/flow-the-psychology-of-optimal-experience-by-mihaly-csikszentmihalyi.html

3. Beyound human-computer interaction
   Interaction design - beyound human-computer interaction 2nd edition, Wiley. (p. 543)

4. Businessinsider.com/blackboard/kinect
   http://www.businessinsider.com/blackboard/kinect

5. Wikipedia.org
   http://en.wikipedia.org/wiki/Time-of-flight_camera

6. Panasonic - D-IMager
   http://www.panasonic-electric-works.com/peweu/en/html/26750.php

7. Structured light 3D scanning
   http://en.wikipedia.org/wiki/Structured-light_3D_scanner

8. Processing.org
   http://processing.org/

9. Processing OpenKinect
https://github.com/nrocy/processing-openkinect

10. Cinder
http://libcinder.org/

11. Cinder Kinect
https://github.com/cinder/Cinder-Kinect

12. Unity programming
http://unity3D.com/unity/engine/programming

13. Mono-project
http://www.mono-project.com/Main_Page

14. NVidia PhysX
http://www.nvidia.co.uk/object/physx_new_uk.html

15. OpenNI
http://OpenNI.org/

16. C# Kalman filter
http://autospreader.wordpress.com/2011/01/17/a-c-kalman-filter-class/
Source code can be found on the appended CD.

17. Kalman filter
A New Approach to Linear Filtering and Prediction Problems,
Kalman, Rudolph Emil. (1960)
Transactions of the ASME--Journal of Basic Engineering, (volume82), p. 35-45.

18. Kalman filter introduction
An Introduction to the Kalman Filter Greg Welch1 and Gary Bishop2
TR 95-041 Department of Computer Science University of North Carolina at Chapel Hill Chapel Hill, NC 27599-3175
Updated: Monday, July 24, 2006

19. Kinect Camera driver
https://github.com/avin2/SensorKinect

20. OpenCV
http://opencv.willowgarage.com/wiki/

21. Prtools
http://www.prtools.org/

22. Arduino
http://www.arduino.cc/