Implementation of natural user interface buttons using Kinect

Matti Ollila

University of Tampere
School of Information Sciences
M.Sc. Thesis
Supervisor: Martti Juhola
May 2013

University of Tampere
School of Information Sciences
Software Development
Matti Ollila: Implementation of natural user interface buttons using Kinect
M.Sc. Thesis, 36 pages, 3 appendixes
May 2013

Abstract

Microsoft Kinect is an add-on component for the Xbox 360 console that enables the use of body gestures and voice commands to control the Xbox device. While majority of Kinect interfaces use full body motions, hand extension type user interfaces also exist.

This thesis attempts to offer alternative means of performing activations in a cursor based hand extension user interface. For that purpose we developed a testing interface containing the two standard controls commonly employed on Kinect's cursor based user interfaces, as well as nine proposed variants of the interface controls, with two of the proposed variant controls being removed during pilot testing as redundant.

The controls were tested on a 20-participant usability study where they were evaluated based on the factors of speed, rate of false activations and user impression. Based on the test results, most of the proposed new controls enabled faster performance but at the cost of a considerable increase in the rate of false activations. While some controls showed promise with further development only the two-handed push button control could rival the low false activation rates of the standard Kinect controls, while simultaneously allowing for much faster activations and being selected by the majority of the test participants as their favourite interaction type.

Keywords: Kinect, human-computer interaction, hand extension, usability study, gestures recognition

Index

1. Introduction

In 2010 the Kinect sensor was published as an add-on component for the Microsoft Xbox 360 gaming console, having sold 8 million units in the first 60 days the device was selected by the Guinness World Records as the fasted selling consumer electronics device [BBC News, 2011]. The device allows for *natural user interface* interaction by tracking the user's movement and location as well as the the use of voice commands to control the Xbox device.

Due to its affordable price, new features and high popularity the Kinect sensor has greatly increased the use of natural user interfaces in gaming. While most Kinect applications track the user's entire body, many do additionally feature a graphical user interface with a cursor that functions as *hand-extension* positioning itself based on the location of the user's hand. Most Kinect hand extension interfaces typically feature one of the two types of controls, the *hover button* control where the user holds the cursor over the control for a set amount of time in order to activate it and the *confirmation hover button* in which moving the cursor over the control reveals an additional confirmation area where the user holds the cursor for set time to activate the control, much as with the hover button control. These cursor based interfaces typically use the users hand for positioning the cursor with little finer detail.

This thesis focuses on developing alternative user interface controls for the Kinect sensor's cursor based user interface as well as performing 20 participant usability test to compare the performance of the new controls to the existing hover button and confirmation hover button controls commonly used in Kinect user interfaces. The custom controls and tests were developed based on the following research questions:

- Could pushing motion gestures be used as an alternative method of activation to standard Kinect user interface hover button controls?
- Could hand gestures be used as an alternative method of activation to standard Kinect user interface hover button controls?
- Could the Kinect user interface's confirmation hover button be made more responsive by eliminating the wait time in the confirmation activation without greatly increasing false activations in the control?

The usability tests were based on three aspects on the control. Firstly, the number of *false activations* performed during the test, where the user performs a different command than expected, or performs a command when none were intended. Secondly, the duration it takes the user to successfully complete the tests on a particular control. Lastly, the impression the user has of the particular control. *The Human Interface Guidelines document* (later referred to as the HIG document) [Microsoft, 2012b] states that the speed of completion is not a factor that should be taken into account when developing Kinect controls, but we felt it was a worthwhile factor to consider and one that could have a negative influence on the user experience if the user felt like the controls took too long to use or interrupted the user's pace.

The new Kinect user interface controls as well as the testing interface were developed, and the tests conducted together with Tommi Pirttiniemi. While this thesis focuses on the practical implementation and testing of the controls, the *Usability of natural user interface buttons using Kinect* by Tommi Pirttiniemi [2012] covers the usability and design aspects of the controls and tests in greater detail.

The Kinect device itself is described in further detail in chapter 2. The featured Kinect user interface controls are described in chapter 3, with chapters 4 and 5 covering the implementation of the user interface side and the Kinect sensor side of the used controls in more detail respectively. Chapter 6 details the design and results of the usability tests. Chapter 7 presents methods and controls that were not featured or were left out of the tests. Chapter 8 elaborates further on the results of the research and covers some opportunities for further development while chapter 9 concludes the thesis.

2. Kinect overview

Our work is based on testing both existing and potential user interface controls for Microsoft Kinect applications. This chapter covers the Kinect device and its features, in particular it focuses on the technical specifications and the common user interface designs.

Kinect is a motion sensing input device developed by Rare, a subsidiary of Microsoft, as an add-on for the Xbox 360 gaming console. Its core functionality relies on technology called ”range imaging” to produce a two-dimensional image with pixel values representing distance instead of colour. Kinect's range imaging system uses a technology developed by PrimeSense for recognizing specific gestures and motions using an infra-red projector, a camera and a special microchip. [MIT Technological Review, 2011]

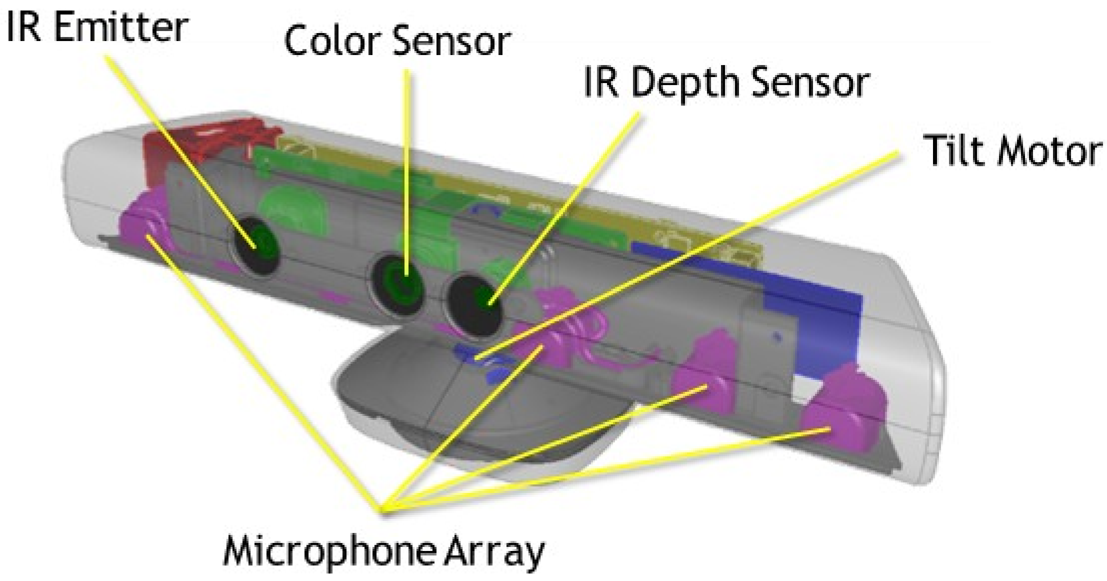

Figure 1. The Kinect sensor and components. [Microsoft, 2012a]

2.1. Kinect specifications

The Kinect sensor consists of several components that can access various types of data, such as an RGB camera, an infra-red emitter and an infra-red depth sensor, which enable the Kinect sensor to record both colour image data as well as depth image data at 30 frames per second. In addition the sensor contains an array of 4 microphones and an accelerometer. This enables the Kinect sensor to not only detect the distance of the objects recorded on the camera, but the location of the recorded sound as well as position of the sensor itself. [Microsoft, 2012a]

The sensor can record data on 57.5° horizontal and 43.5° vertical angle, with additional +27° to -27° vertical angle adjustment available due to a motorized footing on the sensor. The practical distance where a Kinect sensor can accurately record data is from 1.2 meter to 3.5 meter, although a ”near mode” can alternatively be user where the practical use range starts from 0.8 meter, but only reaches up to a maximum distance of 2.5 meter. [Microsoft, 2012b]

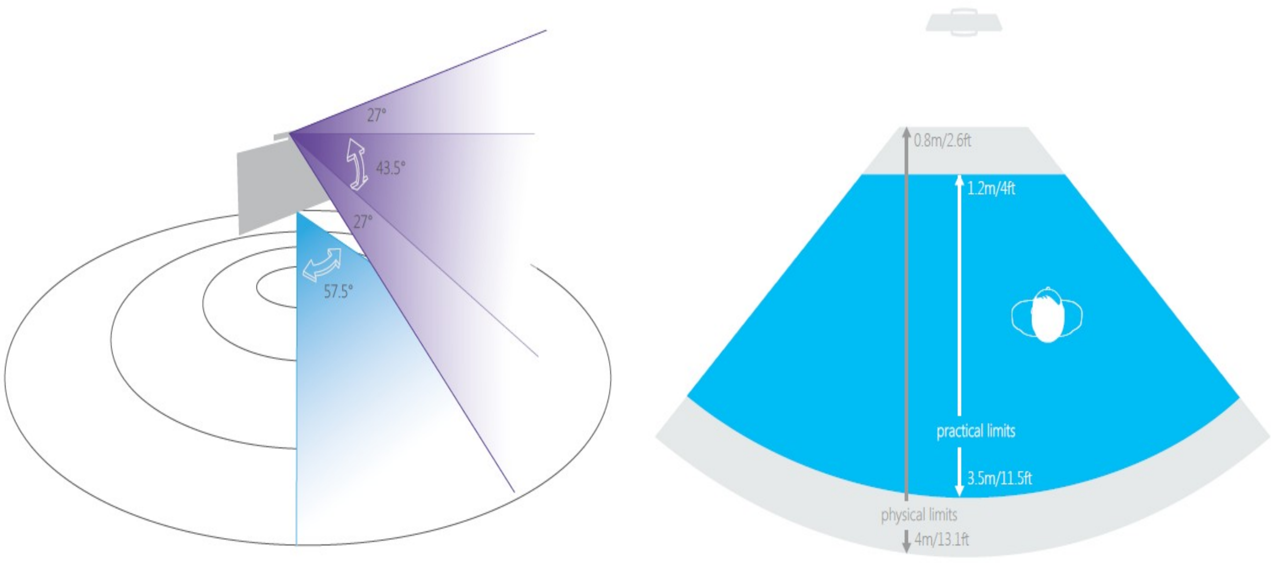

Figure 2. The Kinect sensor range on normal settings. [Microsoft, 2012b]

In addition to the raw sensory data, Kinect sensor can further analyze the recorded data with the ability to recognize and track users within the view of the camera, being able to identify different users, track their positions and provide *skeleton data* of the locations and positions of 20 of the user's joints. According to the HIG document, the sensor has the ability to simultaneously track 6 people, and provide skeleton data on two of the tracked individuals simultaneously [Microsoft, 2012b]. However, Kinect does not currently officially support identifying the user's hand gestures or digit positions [MIT Technological Review, 2011].

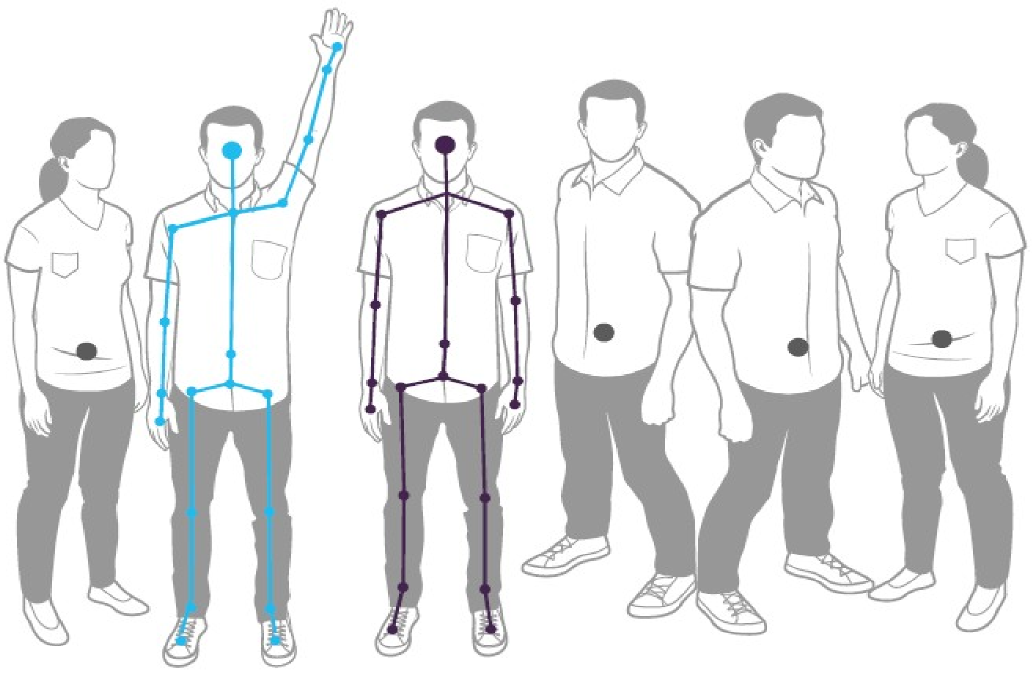

Figure 3. Two instances of skeleton-data and four additional tracked people. [Microsoft, 2012b]

2.2. Common Kinect interface designs

While Kinect's motion tracking is typically used for performing more natural motions using the user's full body, such as dancing, most Kinect-based applications do also feature a more traditional user interfaces, often controlled with hand motions alone. These user interfaces are commonly present in navigating the applications' option screens instead of the main interface itself and often feature a cursor as an extension of the user's hand.

Not all Kinect applications feature cursor based user interface, such as Dance Central [Harmonix, 2010] which uses the user's hand's vertical position to select an item from a simple list and horizontal swipe motions to perform selections. However, in the cases where traditional cursor is used the cursor position typically corresponds to the location of the user's hand, while activations are handled commonly based on the position of the cursor and time – separate activation gestures are rarely ever used.

To perform activations in cursor based user interfaces, a control we refer to as hover button is employed. This control only requires the user to position the cursor over the hover button for a short duration. Sometimes a more complicated control is used, where a separate activation area is revealed when the cursor is moved over the control, and the activation is performed by holding the cursor on the separate activation area instead of the main control. This more complicated user interface control we call confirmation hover button. [Nielsen, 2010]

3. Design of the interface controls

For our usability testing, we developed nine user interface *controls*, some of which were based on existing Kinect user interface controls as well as other potential alternatives not currently used in existing Kinect software. This chapter will describe the nine controls we decided to implement and test as well as the potential technological or usability-related issues in their use.

When developing the controls for Kinect based user interfaces, both the design and the actual programming work naturally divide into two separate *component* categories: *User interface components* and *input components*. Each control created features these relatively separate logical components: on the user interface side the functionality and appearance of the user interface control itself and on the input side one or more methods for recording the input from the user. This could be likened to the separation of the controls presented on the screen and the handling of the mouse cursor on a typical PC graphical user interface. Therefore, the controls, their user interface and input components will be covered separately.

Since we chose to limit our work to cursor based user interfaces specifically, we chose nine user interface controls for our testing: two typical Kinect user interface controls, Hover button and Confirmation hover button, as well as seven custom alternative controls, *Confirmation button*, *Push button*, *Two-handed push button*, *Sticky push button*, *Gesture button*, *Two-handed gesture button* and *Sticky gesture button*. Several of these experimental controls intentionally go against existing guidelines, such as the recommendation for not using separate functions for left and right hand or the inherent difficulties in implementing pushing motions for performing activations [Microsoft, 2012b]. Typical issues the various controls will need to account for are the quality of user experience, the possibility of *false activations* where user's actions are mistakenly perceived as activation events by the system as well as *missed activations* where users attempt to perform an activation is not noticed by the system. [Dix, 2002]. The nine controls themselves are implemented by using one of the five user interface components: *default button UI component*, *hover button UI component*, *confirmation hover button UI component*, *confirmation button UI component* or *sticky button UI component* that are combined with one or more of the three input components: *cursor coordinate tracking input component*, *pushing motion input component* and *hand gesture input component*.

3.1. Hover button

As the default approach in the vast majority of Kinect applications' cursor based interfaces, the Hover button was an obvious choice as one of the controls to be tested. The control is exceedingly simple, requiring the user to do nothing more than hold the cursor over the control for a set time, after which the control is activated [Nielsen, 2010]. The activation is cancelled if user moves the cursor outside the control's boundaries. Our testing components as well as the hover buttons commonly employed on Kinect applications use 1.5 second countdown for the activation.

Hover button's simplicity however is also its Achilles' heel, its time based functionality can rob the user of the feeling of control as the pace is not determined by the users' actions as they are in more traditional interfaces [Microsoft, 2012b]. Furthermore, the timing followed by a 'completed' state of the control lend itself poorly to repeated actions as the user must leave the control's area and return in order to restart the process anew.

Similarly to the control's simplistic usage, the development also required minimal effort; the control uses a custom hover button UI component that reacts to the presence of the cursor, and the only input component needed is the standard cursor coordinate tracking input component required by all cursor based controls.

3.2. Confirmation hover button

When Kinect applications do not use Hover button for their cursor based interfaces they generally use confirmation hover button. The confirmation hover button is very similar in functionality to the standard hover button, though not quite as widely used. Again, as with the regular hover button, the confirmation hover button receives no activation input from user, but simply activates the control if the cursor stays within the borders of the activation area for a set time. The difference to the standard hover button is the requirement of an additional step for the activation: moving the cursor over the control's area reveals a secondary control – one which behaves identically to normal hover button – thus the countdown for the activation only begins when the user moves to the button's area and then relocates the cursor over the additional confirmation area. [Nielsen, 2010]

Using its additional activation step the confirmation hover button strives to decrease the number of false activations compared to the standard hover button and in the process magnifies the hover button's pacing problem further. Repeating the same action is similarly cumbersome as with the hover button, requiring moving the cursor away from the confirmation area – in this case back to the main control – and then back.

Much like the hover button the confirmation hover button also requires only the standard cursor coordinate tracking from the input components, while the UI component uses a special confirmation hover button UI component for the timing based activation functionality.

3.3. Confirmation button

The third of the controls, that use cursor coordinate tracking exclusively, we chose to test was a custom control we decided to call the confirmation button. The confirmation button extends from the concept of the confirmation hover button by excluding the wait time for the activation. Instead moving the cursor over the control's area displays a secondary confirmation area. Moving to this area immediately activates the control, while moving away from the control without going over the confirmation area hides the confirmation component.

By disabling the wait time the control should react at user's pace, helping to preserve the flow instead of interrupting the user with wait times as the other controls relying exclusively on cursor position do. On the other hand disabling the wait times could lead to an increased risk of false activations.

Much like the two hover buttons used in Kinect applications, the confirmation button only needs the cursor coordinate tracking input component, while using a custom confirmation button UI component for the activation logic.

3.4. Push button variants

When using an interface with a cursor that has its position mapped to the x/y coordinates of the user's hand the first intuitive thought for activating the button-like controls is a pushing motion along the z-axis. With push button controls the user positions his or her hand normally over the control and then pushes “down” towards the button in order to activate it. In addition to the intuitive pushing down to activate button with the cursor directing hand functionality we also included two variants to the push button, the two-handed push button which uses the user's dominant hand to position the cursor but uses the user's non-dominant hand to perform the pushing motion for the activation as well as the sticky push button which would accept the pushing motions to the latest control the cursor crossed over, even if the cursor would slip off the control during the activation.

As a user action the push buttons are possibly the most intuitive control in our tests, however attempting the z-axis movement in relation to the Kinect device and the monitor can potentially greatly interfere with the x/y-axis movements used to position the cursor. The two-handed and sticky variants of the push button control were added in hopes of alleviating the interference from the pushing motion.

Since the push button is activated directly through user's actions the control needs to use both the cursor coordinate detection as well as an additional push gesture input component on the input component side to detect the user activations and separate them from all involuntary z-axis movement. With the push motion detection being handled on the input side the UI component side only requires the standard default button UI component, which simply reacts to the coordinates and activation events from the input component side. For the two-handed push button variant same components could be used, with the push motion detecting input component simply tracking the user's non-dominant hand instead of the dominant hand, while the sticky push button uses a separate sticky button UI component that replaces the standard default button to enable the sticky button functionality.

3.5. Gesture button variants

Another potential way to provide activations through user input is through the use of hand gestures. The gesture button is activated through hand-signals gestured by the user when aiming the cursor on the control. The gestures are normally performed by the user with the hand he or she uses to direct the cursor in order to extend the hand extension concept of the cursor based Kinect user interfaces. In addition to the standard gesture button we added the two-handed gesture button variant where the user aims the cursor with their dominant hand while performing activation gestures with their left hand as well as the sticky gesture button where the last component passed over by the cursor will be activated by the user's hand gesture.

The gesture button can potentially allow for variety of actions to be performed in addition to simply activating the targeted component and can support both actions that feel natural to the user as well as actions that are efficient or technically oriented. Unfortunately, performing the gestures can still interfere with positioning the cursor and in addition there are currently technical limitations in the Kinect device that make gathering detailed data such as hand motions difficult enough that the device does not as of yet provide official support to tracking individual digits.

Much like in the case of the push button variants, the gesture button receives its activation events directly from the input components. Therefore, the gesture button uses a hand gesture input component in addition to the standard cursor coordinate detection on the input component side. This again enables the UI side to use the default button UI component for the normal and two-handed gesture button controls as well as replacing it with the sticky button UI component for the sticky gesture button control. As with the push buttons the two-handed gesture button uses the same components as normal, simply having the hand gesture input component track the user's non-dominant hand for activation events.

4. Implementation of the user interface components

The user interface components form half of the Kinect user interface controls as the graphical user interface side functionality. All user interface components function by tracking the cursor position provided by the input component, but depending on the required functionality some components will activate themselves based on the behaviour of the cursor, while others will react to an activation event issued from the input component side with no regard to how the activation event is performed by the user.

Compared to the implementation of the input side components the user interface side components are implemented much like any other cursor based user interface component. Therefore, any Kinect sensor specific implementation is entirely handled on the input side and the user interface components would function equally well with mouse- or touch-based input systems.

This chapter details the implementation of the five user interface components for supporting the nine cursor based Kinect user interface controls; the standard default button UI component for input side activations, the hover button UI component and confirmation hover button UI component for time based activations, the confirmation button UI component for activations based on the cursor movement alone and the sticky button UI component for input side activations based on previous cursor behaviour.

4.1. Default button

Controls that rely on activation events directly from the user through the input components require only a basic button component on the UI side.

The default button UI component is identical to any regular button control featured on cursor based user interfaces, it simply catches any activation events from the input components that are performed when the cursor is within the component's area. At this point a sound effect as well as a pressing down animation are performed and the component is activated.

Figure 4. Default button appearance used in the interface tests.

4.2. Hover button

The hover button UI component expands on standard default button in two ways. Since hover button controls are not expected to receive actual activation events from the input components the component must perform the activation by itself. Furthermore, since the activations are not directly in control of the user it must indicate when the activation is about to be performed in addition to simply indicating a successful activation.

The functionality is accomplished by tracking the movements of the cursor and performing a countdown that automatically performs an activation event as it reaches zero. This countdown is cancelled if the cursor moves outside the component's area. Many hover button controls indicate their timer functionality by animating the cursor as it hovers above the control, while our component animates itself with a gradual filling animation to better match the style of the other controls featured in the tests.

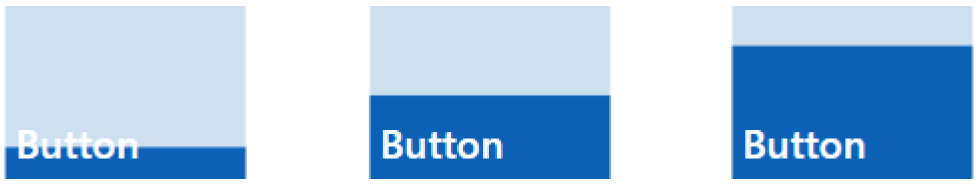

Figure 5. Hover button progression towards activation event.

4.3. Confirmation hover button

The confirmation hover button UI component is a further expansion to the hover button UI component, creating a new area that at first resembles the default button, but immediately reveals a hover button next to the main component area as the cursor is moved over it. This new confirmation area behaves identically to the standard hover button and its successful activation activates the confirmation hover button itself as well as hiding the activation area.

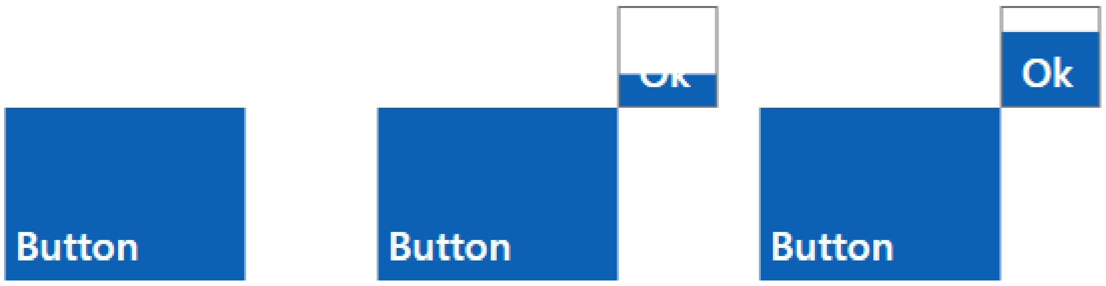

Figure 6. Confirmation hover button progression towards activation event.

4.4. Confirmation button

The confirmation button UI component functions as yet another UI component variant that does not directly receive activation events from user input but instead performs its own activation when the correct conditions are met. Much like the confirmation hover button component the confirmation button component has a secondary activation area that is revealed when the cursor is within the confirmation button's area and hidden when the cursor leaves the area. However, unlike the confirmation hover button there is no activation timer on the confirmation area, but the control is activated instantly as the cursor is moved over a visible confirmation area.

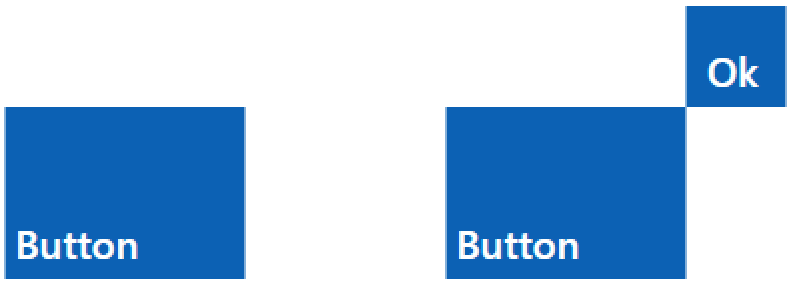

Figure 7. Left: idle confirmation button. Right: confirmation button with the confirmation area.

4.5. Sticky button

While the default button UI component functions with any input component capable of sending activation events and the hover button, the confirmation hover button and the confirmation button UI components replace it for the controls that do not rely on user initiated activation events, the sticky button UI component instead works as an alternative to the default button UI component while still receiving input component activation events without replacing them with activation events of its own.

The sticky button UI component appears identical to the default button UI component and also receives the activation events from the input components, but instead of accepting activation events only when the cursor is within the components area, the sticky button UI components communicate with each other to determine the last component to have had the cursor within the component's area. This latest component will have a different graphical appearance and an activation event will activate the last component regardless of the cursor's current location.

Button
Button

Figure 8. Left: idle sticky button. Right: enabled sticky button.

5. Implementation of the input components

The purpose of this chapter is to describe the implementation of the methods of gathering data on the user's input for a cursor-based user interface. In our work we focused on various methods for implementing a PC-style cursor-based user interface. Kinect does not directly support user interfaces in this manner, but is more suited for larger-scale full body motion input. Therefore, reading the cursor-based input requires a measure of custom implementation.

This chapter elaborates on the three methods of interpreting Kinect data as user input required by the nine user interface methods tested; determining the cursor's coordinates based on the user's posture and confirmation input based on user's depth-wise movement or presented hand gestures.

5.1. Cursor coordinates

The one input component all cursor-based controls require is the cursor coordinate tracking input component. Enabling cursor tracking on Kinect is quite straightforward because of the skeleton data provided by the system, the coordinates on the skeleton data could be applied as the cursor coordinates for a suitable cursor arm-extension on screen.

To allow the user to move about and use the interface from any position on the Kinect sensor's radius the cursor position was mapped based on the difference between the wrist and shoulder points on the user's skeleton data. Furthermore, the central position of the cursor was moved to the side of the user to allow for ease of motion with the hand. Therefore, a right-handed user would center their cursor by holding their right hand to the right and somewhat below their right shoulder, moving the hand to their shoulder to bring the cursor on the left side of the screen and straightening their hand to move the cursor to the right side.

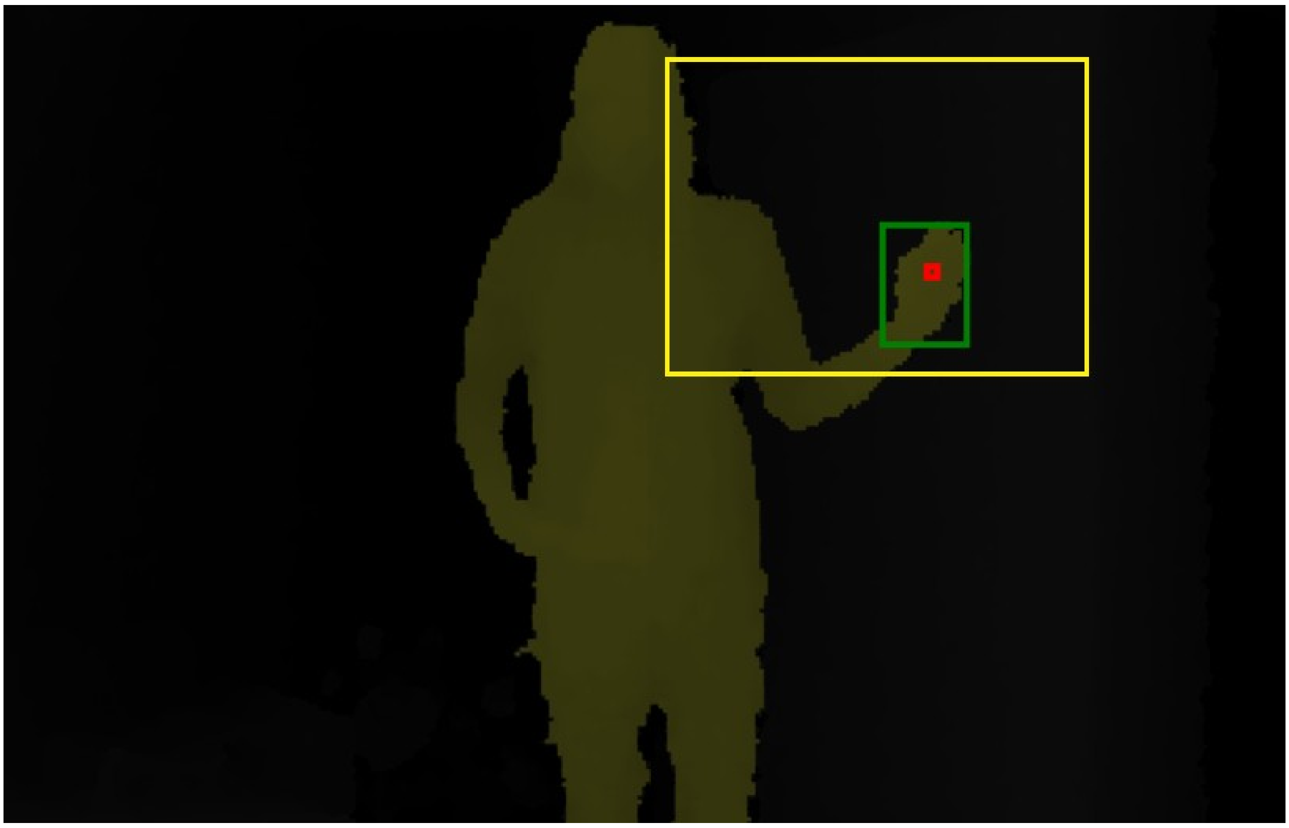

Figure 9. Tracking cursor coordinates. Red dot points the cursor position, the green square covers the active hand and the yellow square represents the cursor area on the user interface.

The data gathered by the Kinect sensor currently has a large amount of interference on it, making the skeleton coordinates jump around erratically, this effect is very disruptive to the user experience so the cursor movement needs to be smoothed by the software before it is applied on-screen. Kinect SDK itself uses the *Holt-Winters double exponential smoothing* [Microsoft, 2012a], that can be used on data in time series form to calculate smoothed data or make predictions [Kalekar, 2004]. Smoothing the cursor values in this way can have a considerable effect on the cursor, but can make it sluggish and unresponsive, causing it to noticeably lag behind the user's actions. The Kinect sensor provides multiple effects for enabling the smoothing and filtering of the data, and the characteristic slow response time of the cursor is generally very noticeable on commercial Kinect applications. Unfortunately, while smooth behaviour of the cursor is integral to the user experience, it can be detrimental to calculations performed for other effects such as matching the skeleton positions to the raw depth data collected from the Kinect sensor. Therefore, we were forced to use very slight smoothing on the Kinect sensor, apply the raw values for our internal calculations and then apply additional slight smoothing of our own when applying data on the cursor position. This proved to be slightly more responsive than most Kinect applications without considerable impact on the accuracy.

5.2. Pushing motions

The pushing motion input component performs similarly to the cursor coordinate tracking component in that it retrieves information from the Kinect sensor's skeleton-data. The concept itself is simple, when user moves their hand on the Z-axis towards the control they are hovering the cursor over, they perform an activation event for the control. However, with the coordinates being tied to the user's hand the Z-axis values are constantly in motion. The changes are even more unavoidable when the user moves their hand on X/Y-axis to position the cursor. Furthermore, what the user views as a pushing motion can vary greatly from person to person, with great differences in the speed and distance of the motion. Therefore, the component needs to be able to detect purposeful variations from the pushing motions while simultaneously avoiding registering the constant Z-axis movement as activation event.

The concept of physically pushing down the buttons on the user interface is extremely intuitive but is technically very difficult to perform. Moving the cursor normally could set off activation events, and performing the pushing motion directly towards the motion without moving the cursor on sideways can be downright impossible for the user.

When attempting to detect the pushing motions, we felt that there were two ways in which the user would attempt to purposefully perform an activation event: high velocity motions and drawn-out motions over long distance, with the variations being in the middle ground between the two. Therefore, we felt that simply setting specific speed or distance limits to the motion would not be enough to account for the variation in user input. To detect the variable range of motions we employed a *push value* that would be gathered over several frames of activity. This value would be adjusted each frame recorded as follows:

$$x = p + s - d\ .$$

Where x is the new push value after the frame, p is the previous push value if one exists or zero, s is the increase in the distance between the user's hand and shoulder and d is the *decay value* variable that can be used to configure how quickly the push value is reduced over time. Once the push value increases past another configurable variable, the *push threshold*, the motion is considered a purposeful pushing motion and the component performs the activation event. If however the decay value causes the push value to fall back to zero or below then the motion is not considered to have been intentional push motion and no activation event is performed.

5.3. Hand gestures

While Kinect provides the “skeleton” data on position and angle of the user's limbs and joints, details such as the placement of the user's digits or the shape of the hand is not supported. Therefore, in order to recognize hand gestures performed by the user we need the hand gesture input component for new, more detailed, functionality.

As previously mentioned, Kinect provides a three-dimensional depth data matrix in addition to the skeleton or the normal RGB camera data. Therefore, recognition of the user's hand gestures is a two-part process of identifying user's hand from the depth data and deducting which, if any, gesture the user is performing.

5.3.1. Reading depth-stream and locating user's hand gesture data

One of the big advantages in reading the Kinect sensor's depth data input compared to other image data is the availability of the skeleton data as well as the ability of the Kinect sensor to distinguish between the users and the background in the depth data as well. Therefore, when we handle the depth data we already have a rough idea which part of the data contains the user's hand and there is no need to run cumbersome pattern recognition algorithms through the entire data, which due to the distortions in the depth stream could easily contain false positives.

With Kinect sensor we can immediately tell if a specific pixel of depth data is part of the background or the user we are looking for, but since unfortunately there is no way to distinguish the hand from the rest of the user further work is required. Since we could gain the rough point of the gesturing hand's location by employing the Kinect SDK's own tools [Microsoft, 2012a] for mapping skeleton points to the depth data and we observed that the depth data behaves practically identically to normal image data with depth coordinates simply replacing the colour values normally present in image data, we decided to employ a highly optimized flood-fill algorithm called *Queue-Linear flood fill algorithm* [Dunlap, 2006] to separate the general area of the user's hand from the rest of the body as well as the background. Because we had the 3-dimensional coordinates of the user's wrist, we could employ a relatively strict limit for filling past areas that were further from the camera than our starting coordinate, but allow for more liberal filling on areas closer to the camera – following the shape of a pointing hand.

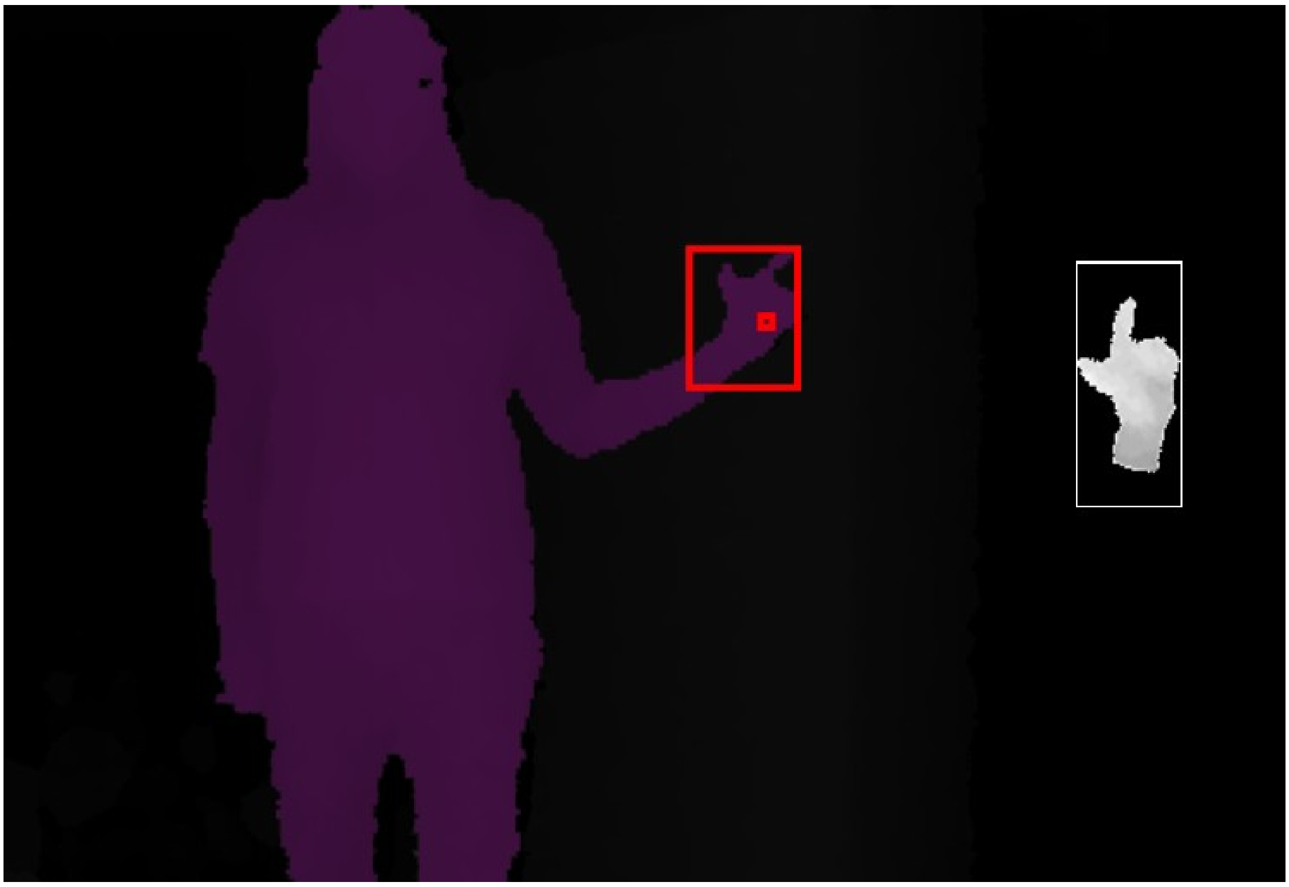

Figure 10. Left: Hand held in the L-shaped activation gesture, red dot showing the cursor position and red square signifying recognized hand gesture. Right: hand gesture data collected and reordered to upright position from 39 degree angle.

After we collected the specific dimensions of the gesturing hand by employing the fill algorithm to the skeleton data's hand location we could simply cut out a smaller depth image that matched the shape and size of the gesturing hand exactly and contained more elaborate data than simply depth with the addition of 'background' identifier being an alternative value to the pixel in place of depth information.

5.3.2. Identifying user's input from the gesture data

After considering several alternatives we felt that due to the additional information available on the Kinect sensor data our system could be more simple and straightforward than most gesture algorithm solutions. Due to the “noise” in Kinect sensors depth data, variation in user performance and the tendency of the skeleton coordinates to vary, we felt that the gesture recognition should use few reliable *keypoints*, much like the ones used in image recognition algorithms such as the *Scale-Invariant Feature Transform algorithm* [Lowe, 1999], but we also felt that due to the need to identify 30 frames of gestures per second and the potential for supporting multiple gestures without greatly increasing the resource consumption, as well as the fact that key information such as the location of the gesture and even the angle of the user's hand were already known, we should instead implement our own *simplified keypoint matching algorithm*.

Since at this point we already have a specific gesture depth data separated from the standard Kinect depth data, and we also have the user's skeleton data which contains both the user's hand and wrist coordinates we can use these points to calculate the angle of the users hands, which enables us to rotate the image to a neutral position – thus enabling a custom setting of either recognizing hand positions in any angle or making them direction-dependant without requiring extra effort from the gesture recognition itself. Most user interfaces, including our test interface, would be more user friendly with the direction-free rotation enabled, but some systems could benefit from maintaining the original direction, such as for example a system for sign language recognition.

With the gesture data already separated from the Kinect input and the hand rotation accounted for, the simplified keypoint matching algorithm splits each gesture image into a matrix of equal dimensions, dividing each gesture image into equal number of areas with each point in the matrix containing the average value of the depth data in a matching area of the gesture image. This enables the algorithm to perform equal comparison of gesture images of varying quality and detail.

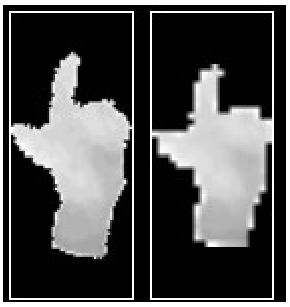

Figure 11. Left: collected high resolution gesture data. Right: resized gesture data.

To learn a specific gesture the algorithm observes a user perform the desired gesture, recording a matrix for the gesture for each frame for several seconds. As each frame of data is being collected the algorithm keeps track of the amount of variation in each point of the matrix. Points with large amount of variation due to interference in the depth data and natural variations in the gesture performed are ignored, while the points that reliably show steady depth data or remain as background are instead saved as *keypoints* for the gesture. When identifying gestures the algorithm can review multiple sets of keypoints for different gestures and compares them against the gesture data recorded from the user. The gesture with highest ratio of its keypoints matching the data in the user's gesture is then selected and if the matching ratio is higher than the *gesture threshold* programmed into the application, the user is assumed to have performed the gesture and an activation event is performed.

When preparing for the usability test, the algorithm was preprogrammed with two variations of a preselected hand gesture, with one variation pointing towards the screen and other performed pointing up as if performing sign language, either constituting a standard activation event. We felt that having each user record their own gestures for the system would be cumbersome during the testing and falsely assumed that the range of comfortable motion and shape of the hand would be relatively uniform with different users.

6. Testing

This chapter elaborates on the design and results of the Kinect user interface control usability tests. For the purpose of testing, a custom testing software was developed, featuring the two most common Kinect cursor based user interface controls as well as seven custom controls. Additional two custom controls were featured but rejected during the pilot testing.

The testing was organized for 20 attendees, with additional two pilot tests for the evaluation of the testing software, environment and testing protocols themselves. The testing conditions were arranged to account for the optimal usage recommendations of the Kinect device [Microsoft, 2012b], with the tests being arranged in a windowless and soundproof area to prevent external conditions such as sunlight and outside noise from interfering with the test results. The testing software ran on Samsung RF711 laptop and Windows 7 operating system connected to a Kinect sensor and video projector that projected a 1024 x 768 pixel view of the testing interface on a screen above the Kinect sensor. The test participant was instructed to stand on the recommended distance of somewhat over two meters from the Kinect sensor, with both the user and floor orientation in full view of the sensor [Microsoft, 2012b]. Environment differing from the recommended usage conditions can be expected to have negative impact on the performance of the Kinect sensor but the possible unfavourable conditions were not included on the tests.

When attending the test the participants were shown the premises and asked to sign a consent form and fill in a questionnaire, as well as being interviewed about their impressions of the components after completing the test. The consent form, questionnaire and the questionnaire results are available as appendixes.

6.1. Usability test design

In order to gather comparable data on the various user interface controls featured, a test interface software was developed. The test software would feature an identical interface for each control type, first familiarizing the user to the tests by running the test on a standard PC graphical user interface with mouse controls and then presenting the tests for the various Kinect controls in a randomly determined order while measuring the user's completion time, correct and false control activations, missed control activations as well as recording the time and location for each activation. Between each control type test the user would be given an opportunity to rest as well as given description of the next control to be tested with an opportunity to test the control before attempting the actual usability tests for it.

While Kinect normally tracks multiple users simultaneously, the cursor based interface would be better served to focus on the active user while ignoring possible distractions. An alternative where multiple cursors are simultaneously controlled by multiple users would not be problematic from a technical standpoint, but was outside the scope of this project. The interface was programmed to activate and focus on a user when one would perform a hand waving gesture in front of the device, the interface would then consider the current user its active user and the waving hand the user's dominant hand while ignoring other users and movement in the camera. The interface would revert to its dormant state to await another user when the current user would either leave the view of the sensor or lower their hands back to a more passive position. However, during the development of the test interface Tommi Pirttiniemi expressed concern that any additional functionality would confuse users unnecessarily and interfere with the testing, so for the purpose of the testing the interface was pre-programmed to constantly focus on the test attendee and the choice of the dominant hand was similarly preprogrammed after pre-test questionnaire would have the user select which hand to use as their dominant hand.

6.1.1. Pilot tests

Before commencing the official testing, two pilot tests were run to analyse the usage of the testing software in practice. The original test program featured a calculator-style interface and 11 types of user controls. In order to mimic a real life use case, the users were tasked to solve simple mathematical calculations with a calculator interface, while the software assigned the user interface control types in random order, each test being preceded with short tutorial on the functionality of the upcoming user interface control.

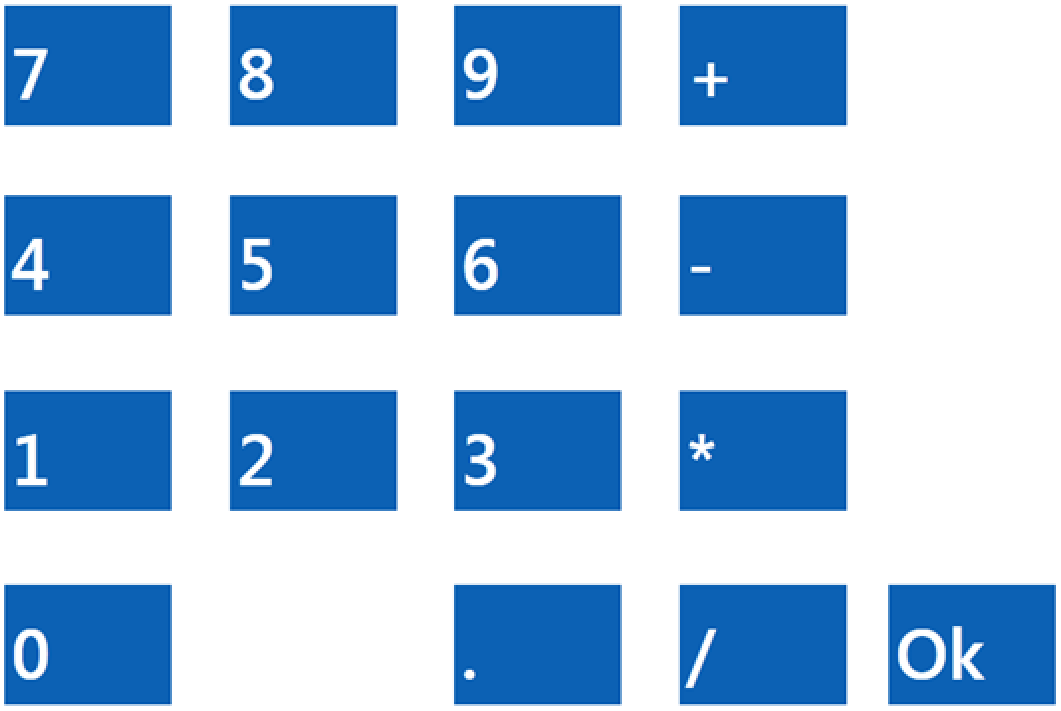

Figure 12. Original prototype test interface.

During the two pilot tests several issues were found and corrected for the upcoming tests. The tests felt too numerous and demanding, noticeably exhausting the people partaking in the test. To solve this two controls were deemed redundant, the *two-handed sticky gesture button* and the *two-handed sticky push button.* Both variants of the existing push- and gesture-buttons seemed too similar to either two-handed controls or sticky controls – containing multiple functions for same purpose and having no perceivable improvement over the normal two-handed or sticky controls already in the tests. Furthermore, the tests themselves not only took too long for the user to finish, but also seemed too confusing as some users seemed unsure if they were expected to use the calculator to perform the calculations or enter the answer after running the calculations in their heads. In order to reduce the amount of work that each test demanded from the user and to simplify the process, the test interface was simplified to a standard numerical pad, while the users were simply prompted to enter the number displayed on the screen.

To provide sufficient feedback each test control on the test was fitted to perform a pressing-down animation when activated, together with the operating system's default mouse and touch screen control clicking sound effect.

6.1.2. Usability tests

With the results gathered from the pilot testing, the testing software was designed to be less taxing for the user and as simple to use as possible. With 9 control types selected for the final testing and the interface simplified to a plain numerical pad with numbers ranging from 1 to 9 and simple instructions for the user to enter the numbers one at the time. The controls in the new testing interface were designed to mimic the simplistic look of the Metro user interface used in the new Microsoft Windows 8 and Windows Phone user interfaces, each control sized 120x85 pixels on the test environment's 1024x768 pixel display.

The testing interface was programmed to first prompt the user for each number once in a random order, after which the user would be prompted to use each control in random order with two-digit numbers that would require the user to use the same control twice in a row, such as 22 or 44. This meant that each of the controls would be tested in various locations of the screen as well as for its ability to accurately receive multiple activations.

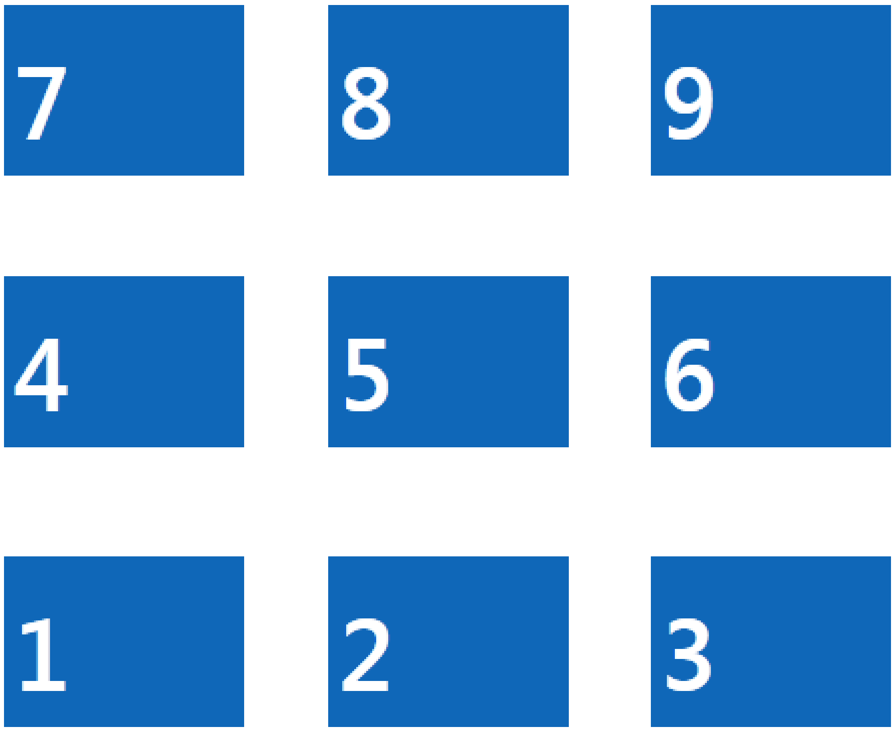

Figure 13. Updated test interface after the pilot tests

6.2. Usability test results

After the initial pilot tests, the proper usability test results were collected from tests run on 20 test participants, of whom none used Kinect device regularly but most considered themselves at least moderately proficient with computers. The controls were measured based on the speed the users would clear the tests as well the accuracy of activating the correct controls. Additionally the users were interviewed on their impressions of the controls.

For testing the gesture buttons, an activation gesture was pre-selected to yield more consistent results and pre-recorded to save time. During the testing it was discovered that some users found it hard to perform the selected gesture, while the accuracy of the gesture recognition algorithm varied wildly from user to user. This lead us to believe that the variance between the range of motion and the general shape of the users hand was much higher than expected, and the hand gesture based controls would have performed considerably better if individually calibrated for each test participant.

To determine the significance of the differences in the measured test completion times we used the Two-tailed t-test [Freund, 1984] to compare the results. Similarly the success rates in activating correct controls were compared using the Fisher's exact test [Fisher, 1922].

The results are presented in detail on the following sub-chapters, with the control names being abbreviated as follows:

C	Confirmation button
CH	Confirmation hover button
G	Gesture button
H	Hover button
P	Push button
SG	Sticky gesture button
SP	Sticky push button
2G	Two-handed gesture button
2P	Two-handed push button

6.2.1. Control speeds

Though emphasis on speed of use was discouraged in the HIG document [Microsoft, 2012b], we felt the speed the user would complete the required tasks was still relevant for the user experience.

The times the test participants took to complete the tests for the various Kinect controls are detailed below.

	Mean	**Standard deviation**	**Fastest Completion**	**Slowest Completion**
C	00:47,100	00:10,052	00:39,000	01:04,000
CH	01:12,500	00:11,700	00:54,000	01:47,000
G	01:12,529	00:52,913	00:32,000	03:39,000
H	00:56,500	00:05,969	00:48,000	01:09,000
P	01:01,550	00:18,594	00:54,000	01:50,000
SG	01:03,105	00:21,561	00:37,000	01:38,000
SP	00:45,850	00:11,609	00:28,000	01:16,000
2G	00:55,300	00:30,802	00:24,000	02:15,000
2P	00:32,850	00:06,780	00:20,000	00:44,000

Table 1. Control test completion times.

To measure the significance of the differences in completion times with the two-tailed t-test [Freund, 1984], seven categories were used in the table below. With NS marking non-significant difference in completion times, +++ standing for $p < 0.001$, ++ for $p < 0.01$ and + for $p < 0.05$

where the component listed on the row outperformed the component listed on the column. Similarly when the column component is considerably faster than the row component --- stands for $p < 0.001$, -- for $p < 0.01$ and - for $p < 0.05$.

	CH	G	H	P	SG	SP	2G	2P
C	+++	+	+++	+++	++	NS	NS	---
CH		NS	---	--	NS	---	-	---
G			NS	NS	NS	NS	NS	--
H				NS	NS	---	NS	---
P					NS	---	NS	---
SG						--	NS	---
SP							NS	---
2G								--

Table 2. Control test completion time difference significances.

Based on speed of use alone, the two-handed push button greatly outperformed the other user interface control types, while sticky push button also outperformed most interface types, both control types performing noticeably better than the standard push button. The gesture button however did not benefit in speed from the two-handed or sticky functionalities in similar manner than the push button did. As expected, the hover button variants commonly used in Kinect applications could not match the usage speed of the new Kinect controls introduced.

6.2.2. False activations

Accuracy in selecting the correct action is crucial on any user interface. Therefore, the user needs to be able to confidently select the intended command, or at the very least be able to notice and reverse a wrong selection quickly. On hand-extension user interface both detecting and reverting the error can often be difficult, so it is important to avoid the false activation in the first place.

The table below shows the amount of false activations on the various user interface controls performed during the testing. Some test participants were unable to finish the hand gesture based tests due the unexpected variation in hand shape and range of motion between people.

	C	CH	G	H	P	SG	SP	2G	2P
Zero false activations	11	20	11	19	14	9	7	14	18
One false activation	4	0	5	0	3	6	7	4	2
Two false activations	3	0	0	0	2	1	3	2	0
Three or more false activations	2	0	1	1	1	3	3	0	0
Interrupted tests	0	0	3	0	0	1	0	0	0

Table 3. Number of false activations in component tests.

The meaningful variations in the amounts of false activations were calculated by comparing the tests that produced *one or more false activations* to tests that produced *no false activations*, with NS marking non-significant difference in the occurrence of false activations, +++ standing for p < 0.001, ++ for p < 0.01 and + for p < 0.05 where the component listed on the row outperformed the component listed on the column. Similarly we used --- for p < 0.001, -- for p < 0.01 and - for p < 0.05 when the column component performed considerably better than the row component.

	CH	G	H	P	SG	SP	2G	2P
C	--	NS	--	NS	NS	NS	NS	-
CH		+	NS	+	+++	+++	+	NS
G			NS	NS	NS	NS	NS	NS
H				NS	++	+++	NS	NS
P					NS	NS	NS	NS
SG						NS	NS	-
SP							NS	---
2G								NS

Table 4. Significant differences in false activation counts.

Reflecting on the Microsoft design focus of accuracy over speed, the hover button variants used in most cursor based Kinect user interfaces outperformed most of the other user interface control types in the tests, as expected. The hover button performed more accurately than the confirmation button, the sticky gesture button and the sticky push button. Meanwhile the accurate confirmation hover button outperformed most controls with only the hover button itself and the two-handed push button as the only new proposed control type being able to match its accuracy. Furthermore, attempting to increase the responsiveness of the confirmation hover button with the confirmation button did lead to significantly reduced accuracy between the two.

The sticky variants of the controls failed to increase the accuracy of the controls on any meaningful amount. However, the two-handed push control did manage to rival the accurate hover button variants in accuracy, but the two-handed gesture on the other hand did not provide enough additional control to make a meaningful difference.

6.2.3. User impressions

In addition to measuring the speed and accuracy of the test participants' performance, we also interviewed the participants on how comfortable they found the controls and attempted to observe whether the participants showed any obvious delight or frustration while using a specific control. The users' opinion of the control is especially important as Kinect applications are typically games and other entertainment. Furthermore, a user is likely to select comfort over good performance if provided an opportunity to select which control they use.

After the tests each participant was asked which control they found most enjoyable as well as which control they enjoyed the least. Below are the number of times a test participant selected specific control as their most or least favorite control or were visibly entertained or annoyed by a specific control. There are more favorites and least favorites than test participants because some test participants would enter multiple controls when prompted for their most or least favorite.

	Favourite	Least favourite	Positive	Negative
Confirmation button	1	1	0	2
Confirmation hover button	0	1	0	2
Gesture button	2	7	1	1
Hover button	3	3	0	0
Push button	1	7	0	4
Sticky gesture button	0	0	0	3
Sticky push button	0	0	0	2
Two-handed gesture button	3	1	0	3
Two-handed push button	13	0	1	0

Table 5. Controls selected as favorite by test participants and controls that provoked test participants.

The two-handed push button was overwhelmingly chosen as the most enjoyable control type to use, likely due to it conforming to the user's own pace while still maintaining the high accuracy as well as its exceedingly simple and intuitive activation gesture. The normal push button was not met with similar enthusiasm, likely due to the difficulty of maintaining the position of the cursor while performing the gesture. The one handed push button holds a shared position of least liked button on the other end of the spectrum from the two-handed push button. The success of the two-handed push button goes against the HIG document's [Microsoft 2012b] rule of not performing alternative functions with left and right hand, but as Nielsen [2010] points out, there are no universal standards in gesture interaction.

The hover button components divided opinions the most, with three people selecting it as the best component, while other three would select it as the worst. The test participants who enjoyed the control appreciated the simple usability and the lack of the need for separate activation command, while others found holding their hand still for extended periods of time uncomfortable and lamented the loss of the ability to activate the controls at their own pace.

The hand gesture button components did delight some participants, largely due to their technological novelty and eloquence of the ability to perform relatively complex commands with minimalistic hand movements. Some participants also found the gestures easy to use and very convenient. On the other hand for others the gestures were difficult, uncomfortable or downright impossible to perform, and the negative feedback for the control was due to the difficulty some test participants experienced in using the control. When performing gestures, two participants had troubles performing the selected gesture, five participants had problems having the gesture recognized and one had troubles over directing the “hand signal” type gestures towards the screen.

Two participants felt that the sticky functionality helped in activating the controls, while one had trouble understanding the concept behind the sticky controls. One participant wished for the ability to relocate the cursor position in relation to the position of the hand. This functionality was in fact implemented as one of the secondary hand signals available for the gesture based components, but was excluded from the tests as only the gesture based components could support multiple different activation types simultaneously. Some test participants complained about the sluggishness of the cursor following the hand, but this feature is equally present in Kinect applications and not a feature of the test software or the controls themselves.

7. Alternative solutions and rejected concepts

Using cursor based user interfaces as arm extension is naturally not limited to the Kinect controls listed in previous chapters. Other controls and methods were considered during the project and of course countless systems and methods have been developed for arm extension user interface both with and without the use of Kinect sensor.

This chapter covers other methods and controls examined during the development, as well as some of the other research into gesture based user interfaces.

7.1. Rejected control types

Due to the modular nature of the proposed Kinect controls, multiple variations were originally considered. While many ideas were ruled out during early planning stages, two controls in particular were only removed during the pilot testing of the software: two-handed sticky gesture button and two-handed sticky push button.

7.1.1. Two-handed sticky push button

Much like the push button variants introduced in chapter three, the two-handed sticky push button was intended as an intuitive activation gesture users would expect on Kinect user interface. As the name implies the control would have user position the cursor with their dominant hand, while performing a pushing gesture towards the button with their non-dominant hand. As with the included two-handed push control, the cursor tracking input component would track the user's dominant hand while the push gesture input component would follow the user's non-dominant hand, additionally the standard default button UI component would be replaced with the sticky button UI component in order for the control to continue accepting activation events even if the cursor would leave the controls area. This was intended to further ease combining the control selecting motion with the activation motion, by not only separating the actions to separate hands but also allowing the actions to be performed at separate times.

However, during the pilot testing phase the testing seemed to feature too many controls for the test attendees to successfully manage. Furthermore, the pilot test results showed no difference compared to the two-handed push button variant, with the users simply activating the control immediately without using the focus maintaining features of the sticky UI button component at all. Because of these reasons the control felt redundant and was removed in order to reduce the number of testable controls to more manageable numbers.

7.1.2. Two-handed sticky gesture button

Very similarly to the two-handed sticky push button above, the two-handed sticky gesture button was a control that combined multiple features to ease combining activation events and cursor aiming, using the sticky UI button component together with the hand gesture input component and the cursor tracking input component to track separate hands and assist keeping the targeted component selected when the cursor movement and activation are not performed simultaneously. The difference to the two-handed sticky push button being only the use of hand gesture tracking instead of push motion.

The component also had very similar results to the two-handed sticky push button during the pilot tests, with the combined functions appearing redundant. The two-handed gesture button was also removed to reduce the overall number of controls in the test.

7.2. Alternative means of identifying gesture data

While Kinect does not yet officially support tracking intricate hand gestures or user's individual digits, there have been many third party attempts at enabling functionality to recognize and track the user's individual digits for more detailed gesture recognition.

During the early phases of the project, various image recognition algorithms were considered and tested on the image and depth data, but were ultimately not used due to their high resource consumption and their limited ability to take advantage of the Kinect sensor's unique features.

7.2.1. Reading hand gestures through digit tracking

Our hand gesture tracking implementation was based on using the Kinect sensor's existing functionality to track the location and direction of the user's hand, and then searching for reliable patterns on the hand gesture performed. A common alternative approach to more detailed hand controls is to instead directly track the user's digits themselves on a similar manner to how Kinect sensor normally tracks the skeleton data on the user's limbs.

Even though Kinect does not officially support digit tracking, several third party developers have implemented their own variants of digit position tracking on Kinect sensor. Notably the Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory (MIT CSAI) [2010] has implemented a Point Cloud Library [Willow Garage, 2010] based solution that individually tracks the location and direction of the user's digits.

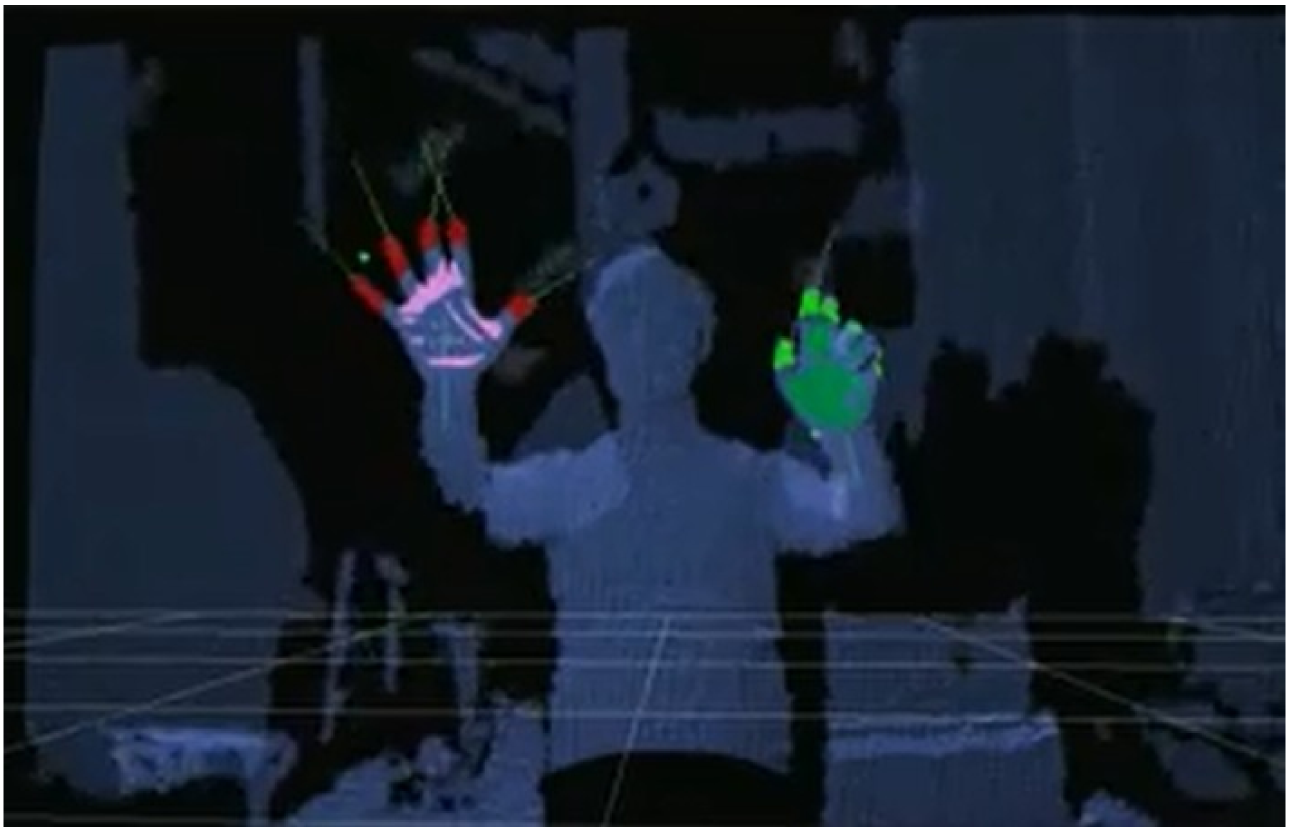

Figure 14. MIT CSAI digit tracking.

7.2.2. Official Kinect SDK hand gesture recognition

Though too late to be accounted for in our project, Microsoft has recently announced a new Kinect expansion to perform rudimentary hand gesture tracking. While considerably more limited than recognizing specific hand gestures or locations of individual digits, the new Kinect SDK versions on Windows platform have been reported to be able to detect whether the user has his hands open or closed. However, there has been no word on this behaviour on Kinect sensor's native Xbox environment. [The Verge, 2013]

7.2.3. Traditional image recognition algorithms

Kinect sensor is of course also able to record standard video image. Therefore, much like with traditional cameras, the various existing methods of computer vision could be employed with Kinect sensor as well. The Kinect sensor's strength, however, lies in its ability to detect distance in addition to the standard video image so a standard image recognition would not take advantage of the Kinect sensor's unique features, such as detecting depth as well as determining the general location of the user's hand outright.

The depth data can fortunately be analysed much like traditional image data, but suffers from the low quality, high amount of noise and limited resources available on the system. Many of the existing methods of image recognition, including the previously mentioned Scale-Invariant Feature Transform algorithm [Lowe, 1999], can be applied to the sensor's depth data as well provided they can perform with the increased noise level in the data. A typical image recognition algorithm would however likely be very resource-intensive. Furthermore, due to the Kinect sensor's ability to track additional information on the user, a normal image recognition algorithm would be performing unnecessary work on already limited resources. An image recognition method that would be able to deal with large amount of noise, consume relatively little resources and able to take advantage of additional information such as the rough position and orientation of the user's hand would however be a likely candidate for hand gesture recognition on Kinect sensor.

8. Discussion and future work

As there is no established standard for gesture based interfaces and due to recent rise in popularity of gesture based systems, especially in entertainment, there is still much that can be done to improve upon the gesture based user interfaces. In our research we proposed various new methods not currently commonly used in gesture based user interfaces and compared them with the existing controls based on speed, accuracy and participant impressions. In this chapter I elaborate on the results of the usability tests as well as the potential for practical applications and further development of the featured controls.

Removing the wait time from the confirmation hover button by using the confirmation button did considerably increase the speed of use compared to the confirmation hover button as well as the hover button, however, compared to both confirmation hover button and the hover button the accuracy of the control was also noticeably affected. One out of our 20 test participants did select confirmation button as their favourite control, but the control was not otherwise any better received than the standard hover and confirmation hover buttons.

Using pushing motions together with the cursor as a hand-extension with the push button and the sticky push button had a high rate of false activations and were poorly received due to the difficulty in combining x/y-axis movement to position the cursor with z-axis movement to perform activation events. On the other hand, the two-handed push button was overwhelmingly popular with the test participants, being selected as the favourite control by 65% of all participants. The two-handed push button also performed much quicker than the hover button and the confirmation hover button without distracting wait times, while still maintaining comparably low rate of false activations.

Activating controls with hand gestures would potentially offer more advanced features and control over a cursor based user interface, but due to limited resources on Xbox environment and the amount of noise in the Kinect sensor data, anything more complicated than the simplistic hand opening and closing functionality provided in the official SDK is likely to perform unreliably. Of all control types the gesture based controls had the highest variation in both completion times and user opinions, likely due to the poor performance of the gesture recognition with specific test participants. Because of the high variation in performance between different test participants and the amount of false activations, the tested pre-configured versions of the gesture based controls cannot be recommended in their current form. However, testing a personally recorded gesture data for each user may potentially yield considerably better results.

In the end only the two-handed push button control could match the low false activation rates of the hover button and the confirmation hover button controls. However, due to the two-handed push button's popularity with test participants and its high performance speed while maintaining the reliable accuracy we find the control to have the highest usability of the controls tested.

There is room for further research on the hand extension alone, many potential control types could not be included in the usability tests and many of the featured controls can be greatly improved if the likelihood of false activations can be lessened.

8.1. Future work on Kinect hand gesture recognition

The hand gesture recognition fared poorly during the testing, but we feel it still offers much potential for the future. Of all the Kinect cursor user interface methods featured, the hand gesture recognition offers the largest variation in activation events, allowing for multiple different commands to be tracked simultaneously. Another benefit of the hand gesture recognition is that it allows for more subtle commands and little space from the user, being potentially more useful while the user is sitting down or has limited space for movement.

Of all the interaction types tested, the hand gesture recognition had the most performance variation from user to user, with some users having great difficulty using hand gestures at all, while others had their best performance and most favourable opinion on the hand gesture based controls. For that reason it is our belief that the low ratings on the hand gesture recognition were largely caused by the activation gesture being poorly selected and pre-recorded based on the developer's own hand. The variance in the range of comfortable hand movement as well as the actual shape of the hand between different users was much greater than originally expected. As some of the test attendees were uncomfortable performing the chosen activation gesture while others had sufficiently different hand shape to lower the odds of recognition, work with more focus on hand gesture recognition could easily afford the time for the test attendee to calibrate to their own hand shape, while simultaneously selecting a gesture they are comfortable using.

While a more forgiving recognition algorithm would have easier time identifying the gestures without specific calibration, the attention to detail on the current implementation allowed for very specific gestures to be performed and recognized. However, the issue could lessen as the the accuracy of the device improves. Alternatively, the gesture recognition could be based on the positions of the user's digits instead of general hand shape as the accuracy of the third party digit recognition on Kinect improves or proper digit-tracking is implemented on the official SDK.

8.2. Future work on expanding cursor based hand extension

While our work was limited to controlling a cursor with the user's hand while performing a single activation gesture to select a control, there are several ways a cursor-based hand extension interface could be expanded upon. For example the interface could track multiple users with multiple separate cursors, or accept multiple separate activation events much in the same way modern mouse-interfaces support various mouse buttons.

In our usability tests the location of the cursor was based on the location of the user's hand moving on an area which had its dimensions defined based on the location of the user's shoulder and the length of the user's reach. However, during the development the hand gesture recognition would recognize additional gestures, such as the ability to move ones hand while the cursor would remain in its previous position. While this feature was not included in the testing, interfaces which allow for multiple types of activation commands could allow the user to shift the location of the cursor in respect to the location of the user's hand, thus allowing for more precision in the movement as well as moving the cursor to further locations than simply the range of the hand. This could be implemented with for example a specific “dragging” gesture, or the distance of the user's hand – moving the hand further away to simulate lifting one's mouse in a mouse-based interface to prevent it from changing the location of the cursor.

Another method of receiving user input overlooked in the tests is the Kinect sensor's ability to detect sounds as well as the direction where the sound originates from. This could easily support sound based functionality, perhaps as additional or alternative activation commands. Further, the commands could be accepted only from the currently active user due to Kinect sensor's ability to determine the location of the sound.

9. Conclusion

To perform the usability tests on the proposed controls, a custom Kinect user interface and testing software was developed, containing the hover button and confirm hover button controls commonly used by cursor based Kinect applications. In addition nine new proposed Kinect controls were designed and developed, with two controls being removed during the pilot testing phase.

After performing the usability tests with 20 participants, only the two-handed push button control was found to match a low rate of false activations comparable to the hover button and the confirm hover button commonly used in Kinect applications. On the other hand, most of the proposed user interface controls allowed the user to perform faster at their own pace while the hover button and confirm hover button controls themselves were not particularly well liked by the users.

Using pushing motions to activate the controls did not perform reliably when coupled with targeting the cursor with hand coordinates and was poorly received by the users. However, performing separate activation gesture as a pushing motion with one hand while targeting the cursor with other easily outperformed other controls in speed and user impression, while maintaining comparable rate of false activations.

Performing activation events with hand gestures did offer interesting prospects, with the possibility of higher range of uses. The hand gestures also had the highest variation in performance, behaving exceedingly well with some while being almost unusable with others. Due to their unreliability between users the gesture based controls had too high rate of false activations to offer a viable alternative but did seem promising for further development.

Removing the wait aspect from confirmation hover button increased the speed of the control, but resulted in unacceptable raise in the rate of false activations.

The results of the tests seemed to indicate that while the hover button and confirm hover button performed very reliably their time-based activation method was disliked by many users. The two-handed push button was the only new control we can recommend at this time. However, the hand gesture based controls did seem promising if they can be developed to perform more reliably.

References

[Dix, 2002] Alan Dix, *Incidental Interaction.* 2002. Retrieved from http://alandix.com/academic/topics/incidental/dix-incidental2002.pdf (1.4.2013)

[Dunlap, 2006] Justin Dunlap, *Queue-Linear Flood Fill: A Fast Flood Fill Algorithm. 2006.* Retrieved from http://www.codeproject.com/Articles/16405/Queue-Linear-Flood-Fill-A-Fast-Flood-Fill-Algorith (11.2.2013)

[Fisher, 1922] Ronald A. Fisher, On the interpretation of χ^2 from contingency tables, and the calculation of P. *Journal of the Royal Statistical Society 85*, 1922.

[Freund, 1984] John E. Freund, *Modern Elementary Statistics, sixth edition*. Prentice-Hall, 1984.

[BBC News, 2011] British Broadcasting Corporation News, *Microsoft Kinect 'fastest-selling device on record'*, 2010. Retrieved from http://www.bbc.co.uk/news/business-12697975 (3.5.2013)

[Harmonix, 2010] Harmonix, *Dance Central*. 2010. Retrieved from http://www.dancecentral.com (6.2.2013)

[Kalekar, 2004] Prajakta S. Kalekar. *Time series Forecasting using Holt-Winters Exponential Smoothing*. Kanwal Rekhi School of Information Technology 2004.

[Lowe, 1999] David G. Lowe, "Object recognition from local scale-invariant features". *Proceedings of the International Conference on Computer Vision. 2.* 1999.

[MIT CSAI, 2010] Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory, *Kinect Hand Detection*, 2010. Retrieved from http://www.csail.mit.edu/videoarchive/research/hci/kinect-detection (13.4.2013)

[MIT Technological Review, 2011] MIT Technological Review, *50 Disruptive Companies*. 2011. Retrieved from http://www2.technologyreview.com/tr50/primesense/ (25.12.2012).

[Microsoft, 2012a] Microsoft, *Kinect for Windows Sensor Components and Specifications*. 2012. Retrieved from http://msdn.microsoft.com/en-us/library/jj131033.aspx (25.12.2012).

[Microsoft, 2012b] Microsoft, *Human Interface Guidelines. Kinect for Windows v1.5.0.* 2012. Retrieved from http://go.microsoft.com/fwlink/?LinkID=247735 (25 .12.2012)

[Nielsen, 2010] Jakob Nielsen, *Kinect Gestural UI: First Impressions.* 2008. Retrieved from http://www.useit.com/alertbox/kinect-gesture-ux.html (6.2.2013)

[Pirttiniemi, 2012] Tommi Pirttiniemi, *Usability of natural user interface buttons using Kinect*. University of Tampere 2012.

[The Verge, 2013] The Verge, *Kinect hand recognition due soon, supports pinch-to-zoom and mouse click gestures*. Retrieved from http://www.theverge.com/2013/3/6/4069598/kinect-for-windows-hand-detection-hands-on (13.4.2013)

[Willow Garage, 2010] Willow garage, *Point Cloud Library*, 2010. Retrieved from http://pointclouds.org/ (13.04.2013)

Appendix 1: Usability test consent form

Please read and sign this form.

In this usability test:

- You will be asked to fill in a questionnaire.

.
- You will be asked to perform certain tasks using Kinect.
- We will also conduct interview with you after the test.

Participation in this usability study is voluntary. All information will remain strictly confidential. The descriptions and findings may be used in our master’s thesis. However, at no time will your name or any other identification be used. You can withdraw your consent to the experiment and stop participation at any time.

If you have any questions after today, please contact Tommi Pirttiniemi at [redacted] or [redacted]

I have read and understood the information on this form and had all of my questions answered

____

Subject's Signature

________________________________ __________________

Usability Consultant Date

Appendix 2: Questionnaire

Purpose:

How well participants can interact with different buttons using only Microsoft Kinect as their input device.

Introductory Questions

- Have your ever used Kinect? ____Often ____Few times ____Once ____Never

- Select your age group ____18 to 25 ____26 to 35 ____36 to 50 ____ over 50

- Do you have any handicap or disability that might affect your arm movements? ____Yes ____No

- How would you rank your computer related skills? ____Beginner ____Intermediate ____Expert

Appendix 3: Questionnaire results

The questionnaire was given to each of the 20 participants, the data should be read “out of 20”.

Have your ever used Kinect?

Often	0
Few times	5
Once	2
Never	13

Select your age group

18 to 25	3
26 to 35	16
36 to 50	1
Over 50	0

Do you have any handicap or disability that might affect your arm movements?

Yes	0
No	20

How would you rank your computer related skills?

Beginner	0
Intermediate	11
Expert	9

The participant's gender was not asked in the questionnaire but it was recorded:

Male	12
Female	8