Universiteit Leiden
The Netherlands

Opleiding Informatica

Creating Models of Interaction for a Video Wall

Cas Dekkers

Supervisors:

Dr. Ir. Fons J. Verbeek & Dr. Kristian F. D. Rietveld

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl 24/08/2017

Abstract

With the emergence of video walls on the global market comes the need of finding intuitive ways to interact with them. This thesis explores ways of controlling video walls with arbitrary human interface devices. It implements logic to create a generic application framework, in which the interaction patterns for these devices are governed. If proven to be usable, the framework can be presented as an open source project and can be extended by adding support for more devices. The ultimate goal of the project is to enable usable interactions of software-controlled video walls with all sorts of human interface devices. That way, future projects and applications can use the framework with their software, allowing for sophisticated interaction capabilities for video walls.

Contents

Chapter 1

Introduction

1.1 Introduction

Video walls are setups of multiple monitors that are tiled contiguously to form a single, larger screen. They can be found mostly in large public venues, such as stadiums and airports, but as the technology becomes progressively more popular, even local news stations and other smaller businesses have one at their disposal nowadays. Data from 2015 suggest the video wall market is likely to have doubled by 2020 [MarketsandMarkets.com, 2015]. The emergence of video walls poses challenges for research into and development of ways for people to interact with them. Their large screen size makes them a type of display device that differs from smartphones, tablets, and desktop computers. Existing Interaction techniques for those devices are therefore generally questionable or not suitable for systems like these. For instance, controlling a video wall with a mouse makes interacting rather unintuitive and unwieldy due to the screen size. This justifies research into finding solutions more befitting video walls.

The primary goal of this project is to find ways of controlling interactive video walls with human interface devices (HIDs). Numerous such devices exist and they differ in the way of capturing and processing human instructions, but they usually share the same purpose of performing certain common actions in a user interface, such as pointing, typing, clicking, or scrolling. This principle provides a basis for the design of a generic application framework, that maps inputs from all sorts of HIDs to a fixed set of such common actions in a user interface. External applications can, and may often already do, make use of these common actions in their functionality. In this way, any software-based video wall that is capable of running applications can be controlled by human interface devices that are supported by the framework.

Because the framework can in theory be used for any video wall that is able to run applications, determining a user group boils down to figuring out what video walls are commonly used for. According to Gek Siong Low, a MSc in Computer Science from Stanford University, video wall apps may include applications in data presentation and visualization, entertainment systems, and the field of e-commerce [Gek Siong, 2003].

Generally, the user group can be described as the group of people that intend to run or create arbitrary applications that are specifically built for being shown on large displays.

In order to test the framework, a video wall application is created that implements such an arbitrary purpose: in this case, a gallery for high-resolution imagery. This secondary task originates from the Bioinformatics department of Leiden University, that has installed its own video wall at LIACS, BigEye, as a means to view large microscopy images without losing the sense of context. The test is meant to determine to what extent a generic application framework for interactions with video walls like this can be usable [Nielsen, 1994].

To get to know the BigEye video wall, this project's first task is to create a virtual remote controller that allows the settings of the displays to be synchronized easily. This part of the project is described in Chapter 2.

The research question of this thesis is: can we design a generic application framework for interactions with a video wall?

1.2 Related work

1.2.1 FreePIE

To clarify the intent of the framework, let us first look at a program similar to it, called FreePIE [Malmgren, 2017].

FreePIE is an open source program written in the .NET software framework that functions as a programmable input emulator – hence, the PIE suffix. What this means is that it can take input from interface devices and translate them to inputs of other devices, after which the latter input is emulated on the target device. For example, FreePIE can be used to control a PC game with a Wii Remote by emulating mouse movement based on the Wiimote's input. Users can write short scripts in a Python-like programming language to tell FreePIE what to do when the Wii Remote's buttons are pressed or how to handle the acceleration data the Wiimote provides. The software addresses devices through libraries that can be added by third-party developers either to the FreePIE core or as a separate library.

The intent of the work in this thesis is to create a generic application framework for interactions with video walls. Not only mice and keyboards can be used for this, but devices such as a Kinect sensor can, too. Like FreePIE, support for multiple devices through open development might therefore be a strong concept for the framework. Frameworks, however, are meant to make application development easier, and for that reason, this project will look for ways to avoid the hassle of having to write FreePIE-like scripts. This is achieved by establishing one basic, fixed gesture set that is provided by the framework instead of by the developer. In that way, developers do not have to be concerned with writing interaction sets and can instead focus on their application design.

1.2.2 Reality-Based Interaction

For researching the creation of a generic application framework for interactions with video walls, a selection of interface devices has to be made to generate gesture sets for. Numerous devices are available, such as smartphones & tablets, game controllers, virtual reality headsets, and gesture-based devices. The devices that are chosen for this project (the Wii Remote controller, the Leap Motion controller, and the Kinect 2, Chapter 3) were picked mainly because each relates to the notion of Reality-Based Interaction [Jacob et al., 2008] to their own extent.

This notion encompasses that interaction styles in GUIs gradually move closer to the real-world. With the ″real″ world, Jacob et al. touch upon naïve physics, body awareness & skills, environment awareness & skills, and social awareness & skills. They state that these themes provide a basis for interaction with computers that comes close to people's interaction with the non-digital world, and that this trend is a positive one. For such reality-based interactions, people are theorized to already possess the required mental skills needed to operate a certain system, which might speed learning. In situations with pressure due to time, stress or information overload, this interaction style may improve performance. RBI may also encourage users to improvise and explore because they don't have to learn interface-specific skills.

Looking at the selected devices, it can be shown that the gesture sets they were designed for gradually become more ″real″. The Wii Remote is a simple pointing device, almost like a TV remote control, with the addition of pointer-tracking capabilities. This last feature makes it a bit more realistic than that, though, since moving the remote around in space affects the pointer position on a screen. Leap interactions are more realistic: there is no external controller other than your own hand. The Leap can track the motion of an entire hand or arm with extreme precision, which provides for physically grabbing, waving, pointing, and more to make an interface respond to a user's actions. The Kinect 2 does not only track the hands or arms, but the entire body. When standing in front of a life-sized screen, these interaction patterns can make interfaces respond with realistic, full-body gestures.

It is not enough, however, to only make the interactions or the interface as reality-based as possible in order for a design to be successful. According to Jacob et al., developers have to find a balance between including unrealistic, but necessary elements and reality-based parts in their UI. They state that many designers make these balancing decisions implicitly, and that RBI makes such tradeoffs explicit instead. It provides explanatory power for understanding the costs and benefits of such decisions. Therefore, during the design phase of this application framework, the notion of RBI is used as guidance to developing a balanced interface.

During the test of the application framework, the effects of RBI can be measured to see whether it possesses the acclaimed positive influence on this project's implementation.

1.3 Definitions

This section serves as an overview for the definitions of common terms in this thesis.

BigEye

The video wall setup at LIACS, which is used for developing the framework described in this thesis

Generic application framework

A fundamental basis that external applications can use as a foundation for particular tasks; the framework is reusable and has the purpose of making the development of a specific class of applications easier

Graphical User Interface (GUI)

A user interface that utilizes icons and visual indicators for navigation

Human Interface Device (HID)

A device that, when connected to a computer, is able to capture and process human inputs and provide some output based on that information

Kinect

Refers to the sensor (HID) or software developed by Microsoft for use with their line of Xbox video game consoles; the Kinect sensor has motion and joint tracking capabilities, which the Kinect software can process to enable full-body gesture recognition

Leap Motion controller

A sensor (HID) built for tracking hand motion and gestures. It provides precise hand, finger, wrist, and arm data for application development

Software-based video wall

A video wall that is attached to a computer capable of running third-party applications through an operating system

System Usability Scale (SUS)

A questionnaire for users testing computer environments or applications, designed to reliably measure usability

Usability, usable

Measurement of the learnability, efficiency, errors, memorability and user satisfaction with which predefined goals can be achieved by users in a certain environment [Nielsen, 1994]

User Interface (UI)

The means through which a human controls and interacts with a computer or application

Video wall

A setup of multiple monitors tiled together to form a single, larger screen

Video wall application

An application that is designed to run on a video wall in full-screen

Wii Remote or **Wiimote**

A remote controller (HID) with pointer tracking capabilities, designed by Nintendo for use as the primary controller for their Wii video game consoles

1.4 Thesis overview

This bachelor thesis is part of a project at LIACS and is supervised by dr. ir. Fons J. Verbeek and dr. Kristian F. D. Rietveld. The remainder of this work has the following structure. This chapter contains the introduction, related work, and terminology that might not be immediately clear to every reader; Chapter 2 discusses the creation of a virtual remote controller for BigEye; Chapter 3 reports the setup and devices used to explore the research question; Chapter 4 is concerned with the design of the application framework and the gallery for high-resolution images; Chapter 5 touches on the implementation of the framework and the gallery; Chapter 6 contains the experiments that have been performed to test the method used; and finally, Chapter 7 presents conclusions based off of the experiments.

Chapter 2

Creating a Virtual Remote Controller

2.1 Introduction

As part of getting to know the video wall at LIACS, a virtual remote controller for the displays is implemented. This makes it easy to apply general display settings, such as brightness, contrast, or volume, without having to use a physical remote for every display individually.

2.2 Method

The displays of the video wall are Philips BDL4777XL's. They can be interconnected with serial cables and communicate through the RS232C protocol [Electronic Industries Association. Engineering Department., 1969]. A Serial Interface Communication Protocol [Koninklijke Philips N.V., 2011] is available according to which the communication can be modeled.

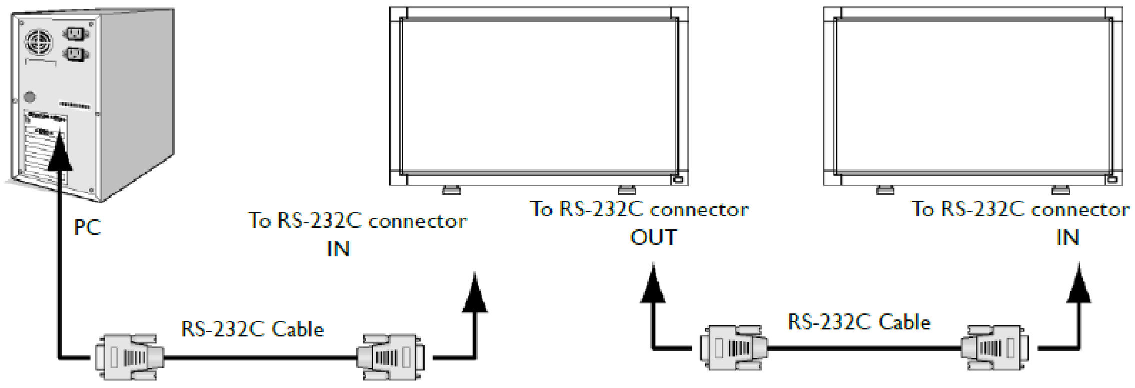


Figure 2.1: Interconnection of the PC (host controller) and two of the video wall displays.

Source: [Koninklijke Philips N.V., 2011]

The protocol relies on sending and receiving packets. They are sent from a host controller – in this case, a PC running Ubuntu 16.04 – and are received by each of the displays, which can respond by sending a package in turn. The displays and the host controller are interconnected in the way shown in Figure 2.1. Packets serve to send commands to displays, such as adjusting brightness, but they are also used to provide the host controller with display data, such as the volume level.

A packet of the RS232C format is a sequence of at most 40 bytes (Figure 2.2) in a hexadecimal format. Each byte value represents a field in the packet and thus ranges from `0x00` to `0xFF`.

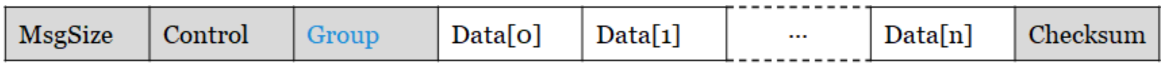


Figure 2.2: The RS232C packet format.

Source: [Koninklijke Philips N.V., 2011]

- The `MsgSize` field contains one byte, representing the number of bytes, or the length, of the packet. Its value ranges from 3 to 40 (`0x03` to `0x28`).
- `Control` is a byte that contains the value of the display that the packet should address. It is often referred to as the monitor ID, as each monitor has its own unique display address
- All displays have a monitor ID, but they can also belong to a monitor group with a certain ID. If the `Group` field of a packet is set to an nonzero value, all displays of which the group ID has that value will be addressed.
- The `Data` fields specify a command code and data to send or to query. The `Data[0]` byte specifies what data to get from the display, or set the display to. For example, when `Data[0]` is set to `0x19`, and the rest of the `Data` fields are empty, the packet requests the display to report its current power state. The other `Data[]` fields are optional parameters. At most 36 (`0x24`) `Data` fields can be present in total.
- The `Checksum` field is meant to verify the integrity of the packet. It is constructed by taking the exclusive `OR` of all bytes in the message, except the checksum itself. If the checksum field of an incoming packet does not have the expected value, an error during transmission might have occurred, and the packet should be discarded.

This packet format allows for getting data from or setting data of the displays. The RS232C protocol provides for two separate flows from the host controller to the monitors and back in these cases. These are detailed in Figures 2.3 and 2.4.

If the host controller initiates a get command, say, by requesting the power state (get command `0x19`) of the monitor with display address 1, according to the SICP, it should send the following packet:

```
MsgSize   Control   Group   Data[0]   Checksum
0x05      0x01      0x00    0x19      0x1D
```

After the monitor has received the packet, it takes some time to generate a response packet to the request (Figure 2.3). If the request is currently not supported, or if it is not valid, the monitor responds to it by sending either a `NAV` (not available) or `NACK` (not acknowledged) packet, respectively. In any other case, the monitor responds with a packet containing the requested information. In this case, it might respond with

MsgSize	Control	Group	Data[0]	Data[1]	Checksum
0x06	0x01	0x00	0x19	0x02	0x1C

meaning the monitor with display address 1 is currently powered on (`0x01` means powered off, `0x02` means powered on).

This process of responding to the request with the appropriate data report can take up to half a second. If no response is received from the monitor within that time frame, either the request packet or the response packet might have been lost in transmission, indicating the request should be sent again.

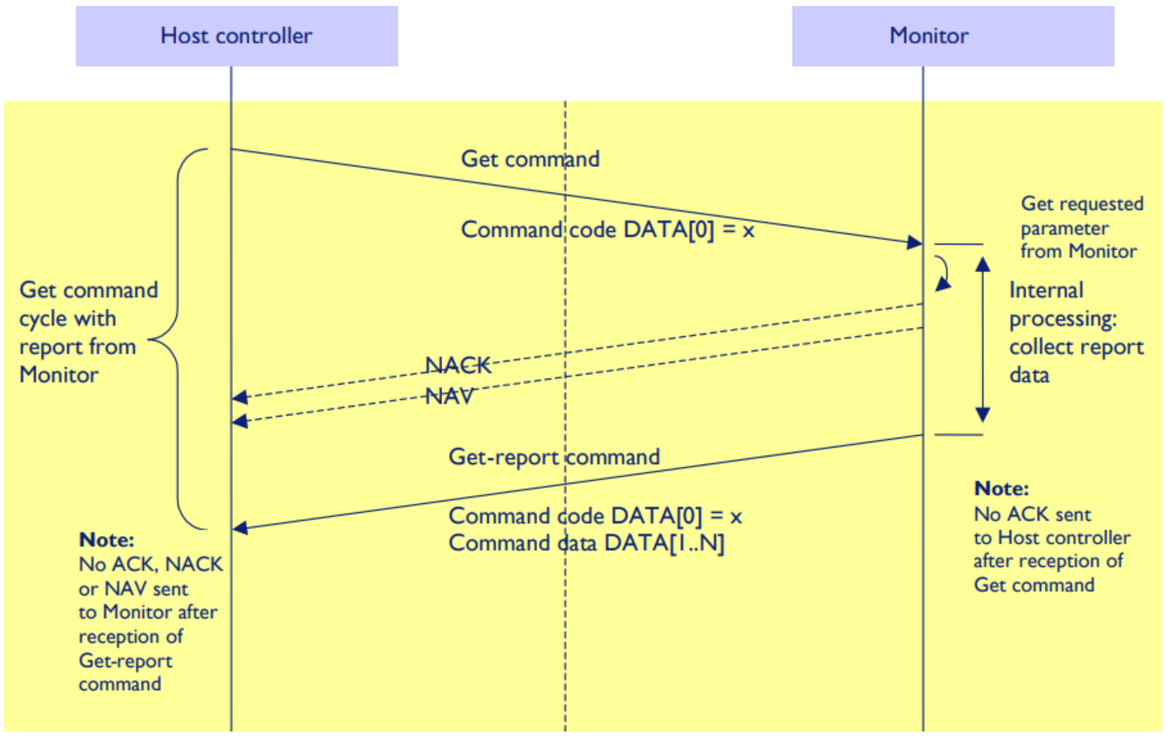

Host controller
Monitor
Get command
Command code DATA[0] = x
Get requested parameter from Monitor
Get command cycle with report from Monitor
Internal processing: collect report data
NACK
NAV
Get-report command
Command code DATA[0] = x
Command data DATA[1..N]
Note: No ACK sent to Host controller after reception of Get command
Note: No ACK, NACK or NAV sent to Monitor after reception of Get-report command

Figure 2.3: Detailed packet flow between the host controller and a monitor in case of a get command.

Source: [Koninklijke Philips N.V., 2011]

In case the host controller sends a set command, the packet flow between the host controller and the monitor is somewhat different (Figure 2.4). The main reason for that is because set commands require no data response from the monitor: these commands simply set some properties of the display. They are simply responded to by sending a `NAV` (not available), `NACK` (not acknowledged), or `ACK` (acknowledged) packet back to the host controller. A thing of note is that set commands of which the `Group` field has been set to a nonzero value do not initiate either of these responses.

For example, let's say the host controller wants to turn off (set command `0x18`) the monitor with display address 1. The following packet is sent:

MsgSize	Control	Group	Data[0]	Data[1]	Checksum
0x06	0x01	0x00	0x18	0x01	0x1E

Given the packet arrives at the monitor, this will initiate one of the responses with command code `0x00`, indicating a generic report message:

- In case the command is well-executed, an `ACK` is returned (`0x06`):

MsgSize	Control	Group	Data[0]	Data[1]	Checksum
0x06	0x01	0x00	0x00	0x06	0x01

- In case the command is not acknowledged, a `NACK` is returned (`0x15`):

MsgSize	Control	Group	Data[0]	Data[1]	Checksum
0x06	0x01	0x00	0x00	0x15	0x12

- In case the command is not available, not relevant, or if it cannot be executed, a `NAV` is returned (`0x18`):

MsgSize	Control	Group	Data[0]	Data[1]	Checksum
0x06	0x01	0x00	0x00	0x18	0x1F

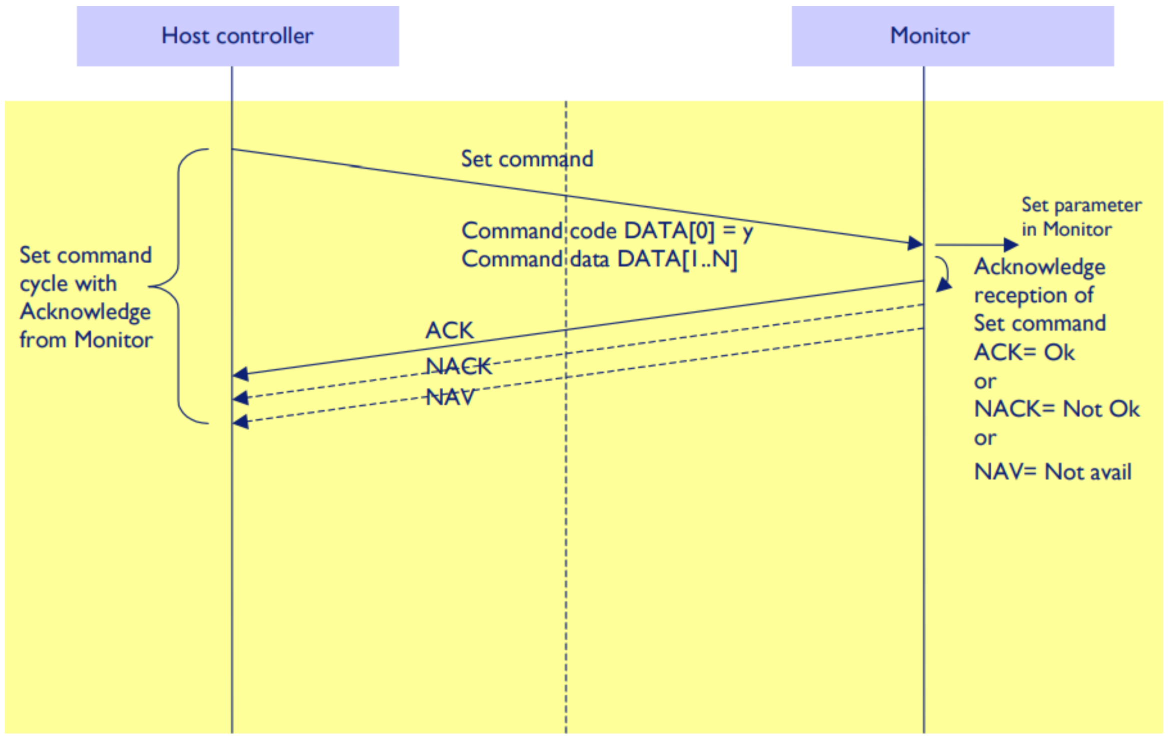


Figure 2.4: Detailed packet flow between the host controller and a monitor in case of a set command.

Source: [Koninklijke Philips N.V., 2011]

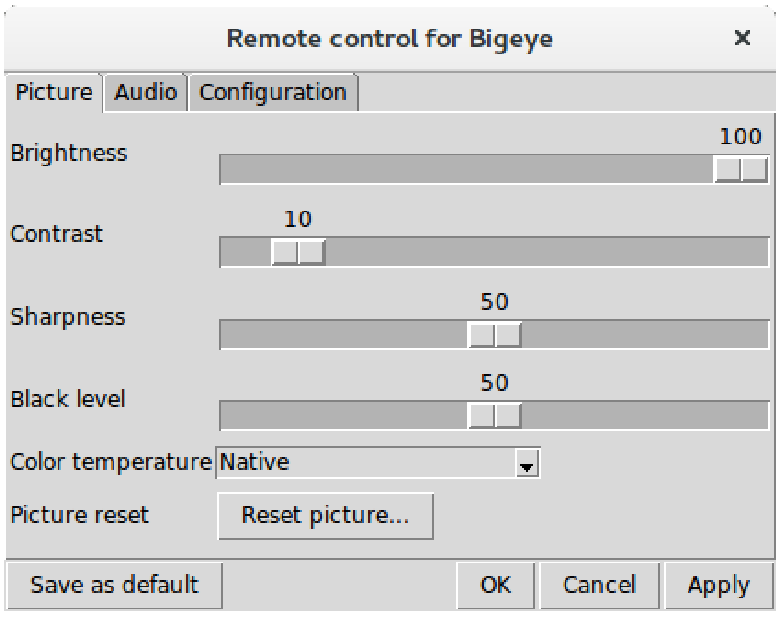
2.3 Design

The end user of the remote control application is mostly interested in only a subset of all available display functions. That is, the functions that synchronize the appearance and behaviour of all the video wall's displays. The following were picked out of all available programmable functions:

- Video: brightness, color, contrast, sharpness, color temperature
- Audio: balance, bass, treble, volume, volume limits, mute
- Configuration: start-up logo, switch-on-state

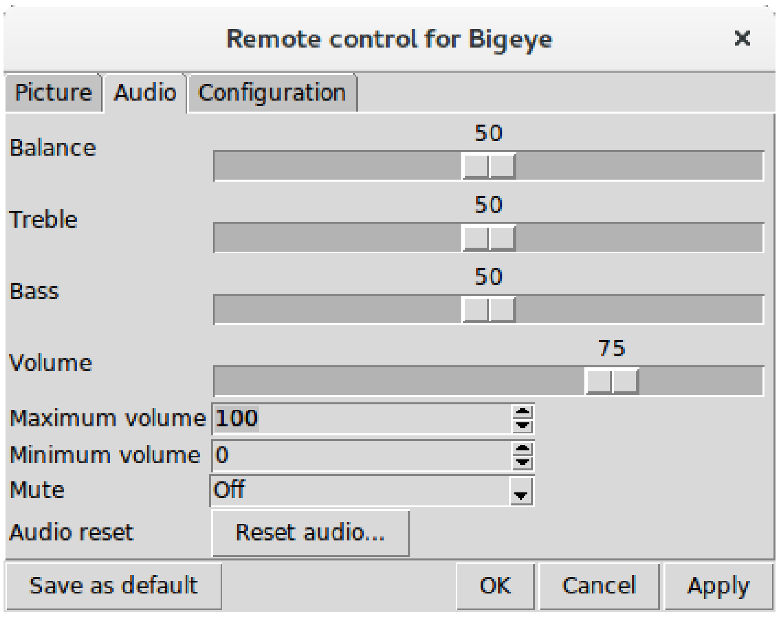
Being able to set these functions in a GUI is easy as opposed to being required to figure out the ”hidden” functionality of a command-line interface. For that reason, the following GUI has been created (Figures 2.5, 2.6, 2.7). The application features some often used elements:

- A tab layout, for separating the functionality of the remote by category;
- Sliders, spinboxes, comboboxes, and buttons that are appropriate for the nature of the functions;
- An `OK/Cancel/Apply` dialog for persisting the settings to all the displays at once.
- Additionally, the GUI includes functions for setting some values as default, so that users can restore preferred settings values easily.


Remote control for Bigeye

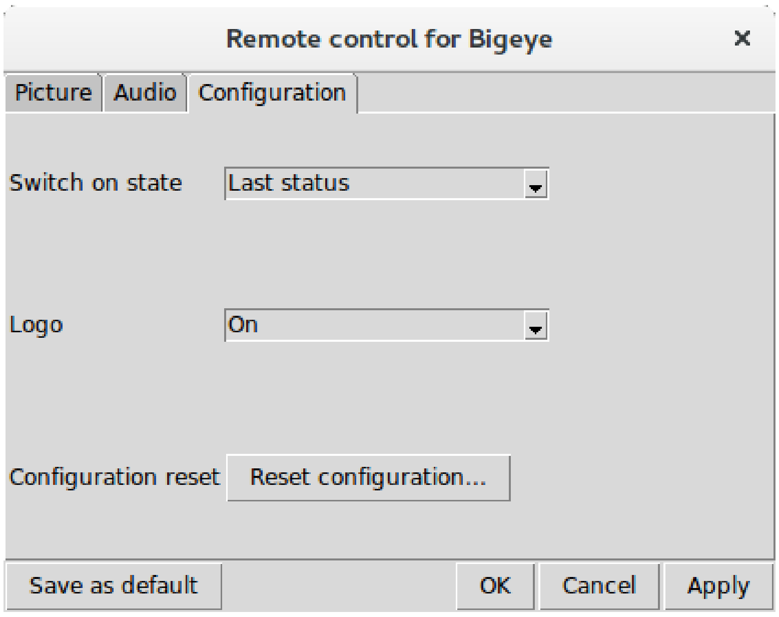
Brightness 100
Contrast 10
Sharpness 50
Black level 50
Color temperature Native
Picture reset

Figure 2.5: The UI of the remote control, showing a tab for video settings.



Balance 50
Treble 50
Bass 50
Volume 75
Maximum volume 100
Minimum volume 0
Mute Off
Audio reset Reset audio...

Figure 2.6: The UI of the remote control, showing a tab for audio settings.



Switch on state Last status
Logo On
Configuration reset Reset configuration...

Figure 2.7: The UI of the remote control, showing a tab for configuration settings.

2.4 Implementation

The functionality for the serial communication is implemented using Python 2.7. Besides several core packages, it relies on the following libraries:

- pySerial 3.4 [Liechti, 2017] for managing the serial connection;
- Tkinter [Lundh, 1999] for GUI programming;
- ConfigParser 3.5.0 [Langa, 2017] for maintaining default settings.

Chapter 3

Materials & Methods

In this chapter, the setup and context regarding the development of a generic application framework for video walls and its test application are introduced.

3.1 Setup

For this project, the software-based video wall setup available at LIACS, called BigEye, is used. It has 12 displays that are arranged in a 3×4 grid of 47″ monitors, with a diagonal totalling 154″ at a resolution of 5760×4320 pixels. Attached to it is a powerful desktop PC, featuring three high-end graphics cards. Each card manages the displays of one of the three columns and thus controls four monitors. The system runs both Ubuntu 16.04 and Windows 10.
For implementing the functionality of the virtual remote controller, which is set out in Chapter 2, the displays' serial ports are used. Each display has an input port that accepts and an output port that passes through serial data according to the RS-232C protocol [Electronic Industries Association. Engineering Department., 1969]. The serial input originates from the output port of a separate machine running Ubuntu 16.04.

3.2 Human Interface Devices for interaction

There are numerous Human Interface Devices (HIDs) out there that are suitable for establishing video wall interactions, but the following were chosen due to their relation to the notion of Reality-Based Interaction [Jacob et al., 2008]. This concept is explained in Section 1.2 (Section 1.2.2).

3.2.1 Kinect 2

The Kinect 2 is a sensor that can track body motions and recognize gestures of up to 6 people simultaneously. The device, which is shown in Figure 3.1, was originally developed for interacting with Microsoft's Xbox consoles. It is equipped with a 1080p color camera and a depth sensor, both capturing up to 30 frames per second; an infrared (IR) emitter; and a multi-array microphone. These features can be used to process data of people's joint positions, physical actions and sounds in three-dimensional space. Usage of the Kinect 2 enables full-body gesture recognition for implementing interactions with the video wall. The use of the whole body for life-sized video walls as a means of interaction is theorized to prove beneficial if designed according to the notion of Reality-Based Interaction [Jacob et al., 2008]. Other benefits of utilizing the Kinect 2 include an extensive Software Development Kit (SDK) for Windows and third-party libraries for several Linux distributions. The SDK includes libraries and gesture recognition software supported and maintained by Microsoft, that make application development relatively easy. Finally, parts of the design phase for an application for a video wall might be based upon or extend the way Microsoft has implemented gesture recognition and user interface design for their consoles.

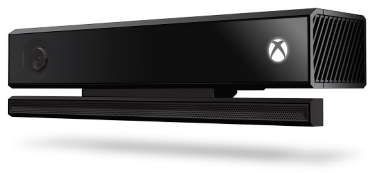


Figure 3.1: The Kinect 2 sensor.

Source: [Microsoft, 2017]

3.2.2 Leap Motion controller

The Leap motion controller is a sensor that tracks hand, finger, wrist, and arm movement. It is the small peripheral device in front of the computer in Figure 3.2. It features two monochromatic infrared cameras and three infrared LEDs, and was originally designed to be placed on a tabletop, facing upwards. With these specifications, it is able to capture extremely precise hand motion [Weichert et al., 2013] at up to 120 frames per second. Recent global use of the Leap focuses on visualization and interactions for virtual reality headsets. It provides a gesture set of detailed hand motions for use with video walls.

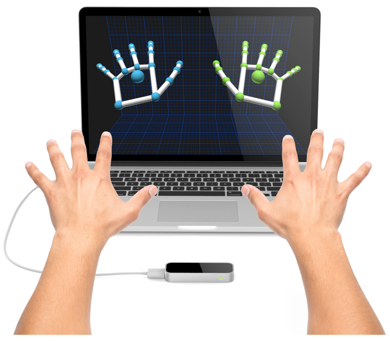


Figure 3.2: Visualization of hands on a PC using the Leap Motion controller.

Source: [Lumo Interactive, 2016]

User interfaces for the Leap can be designed according to the notion of Reality-Based Interaction [Jacob et al., 2008]. Implementations of just hand gestures might prove to be more subtle and perhaps more usable than full-body gesture implementations, such as for the Kinect 2.

3.2.3 Wii Remote controller

The Wii Remote controller (Figure 3.3) was designed originally for use with Nintendo's Wii console (2006). It is a wireless device that communicates through the Bluetooth protocol. The controller sends acceleration data of the device along three axes, along with data of capturing up to four infrared points. The latter functionality is intended for maintaining a reference point of a display, and implicitly the remote itself, in 3D space. Another device, called a sensor bar, is supposed to be placed centrally above or below the display and provides for two of these infrared points. This allows for a pointer to be displayed on screen at the intersection of the remote's pointing axis and the display.

Figure 3.3: The Wii Remote controller.
Source: [My Nintendo News, 2013]

The Wii's UI consists mostly of large block icons that can be selected and clicked on by pointing the Wii Remote to the icon on screen and pressing its buttons. It has been an immensely popular console [Thorsen, 2010] and many people are therefore familiar with its point-to-click concept. Functionality for it is implemented in the framework and can also be tested according to the notion of RBI [Jacob et al., 2008].

3.3 Development environment

The performance of a software-based video wall relies heavily on the balancing of its system's capabilities. When choosing a development environment for the application framework, it is therefore important that the method chosen is capable of maintaining acceptable performance, regardless of the system's exact specifications. For that reason, it is desirable to use a high-level programming language that can manage system resources efficiently.
The languages Java, Python and Visual C# were picked during preliminary tests for reading and processing data from the Wii Remote using several third-party libraries. Visual C# with Microsoft's Visual Studio IDE [Microsoft, 2017] was capable of running the tests flawlessly. Additionally, Visual Studio provides for advanced software development features that suit the time-bound nature of this project. It was picked over Java and Python, which were both inconsistent in managing the Bluetooth connection with the Wii Remote.

The following libraries are used for managing the devices selected in section 3.2:

- Kinect for Windows SDK 2.0 [Microsoft, 2014a]
- Leap Motion Orion 3.2.0 [Leap Motion, Inc., 2016]
- WiimoteLib v1.7 [Peek, 2009]

The framework is tested by running a separate application on top of it. This application represents an arbitrary video wall app that utilizes the core functionality which framework provides. As mentioned briefly in the introduction (Chapter 1), it takes the form of a gallery for high-resolution images. The functional code for the gallery is written in PHP and the GUI is written in HTML, CSS and Javascript. This combination provides for writing an effective GUI with a relatively short development time. The application is executed on BigEye.

For clarity, the following programming environments and libraries are used for creating the gallery application:

- XAMPP for Windows 5.6.31 [Apache Friends, 2017], which allows a Windows machine to run a local PHP server for building web applications;
- libvips [Cupitt, 2017], a command-line tool that is used to divide a high-resolution image into smaller tiles – more on this in Section 5.2;
- OpenSeaDragon 2.3.0 [CodePlex Foundation and OpenSeadragon contributors, 2017], an image viewer written in Javascript that can serve tile source images efficiently;
- dragscroll 0.0.8 [Prokashev, 2016], a Javascript microlibrary that changes allows for scrolling in a browser window by clicking the left mouse button and moving the cursor.

Chapter 4

Design

4.1 A generic application framework for video walls

4.1.1 Core features

Creating an application framework means providing a foundation for external applications to build their functionality upon. One its most important advantages is that it should make writing a certain class of applications easier. It provides developers with a way to deal with an abstraction of the problems or features their applications are designed to solve or implement, without the need of having to spend time and resources for that themselves. This abstraction, however, can also be disadvantageous if a developer has a desire for functionality that is too specific for the framework to provide.

This section discusses the design of a framework for interactions with video walls. The framework's purpose is to make it easier for developers of video wall applications to deal with end users interacting with their apps through the video wall. To achieve this, the framework provides some basic functionality through the following abstractions.

- Firstly, the framework is meant only for software-based video walls – that is, video walls that have the ability to run third-party applications through an operating system. This is a requirement because both the framework and the applications can only be run from within an operating system.
- Secondly, the framework is built on the assumption that most of a video wall application can be controlled by using a specific, limited set actions for a mouse and keyboard.
- Finally, applications that are built upon the framework are assumed not to deal with the inputs of HIDs other than the mouse and keyboard for interactions with their UI.

4.1.2 Functional design

The core idea of the framework is to capture inputs from arbitrary HIDs and provide output in the form of emulating certain mouse and keyboard actions. These actions can be used to achieve certain goals in the UI of an arbitrary video wall application that is built upon the framework. Since this may sound rather abstract, the following simple scenario is given as an example.

Consider a video wall application, called A, that is built upon the framework presented in this thesis. An end user, who is in control of the video wall setup, can interact with the system by using the mouse and keyboard. He opens A and can start interacting with it by, say, clicking and typing around. A is fully functional and does not require the use of the framework.
At any point in time, the end user can switch from using the mouse and keyboard to using a different HID by starting the application framework and selecting the device of choice. The framework is presented as a dialog in which the end user can pick from a list of devices that are available to him. Let's say he chooses to switch from using a mouse and keyboard to a Wii Remote. He opens the application framework with the mouse, which allows him to connect to the Wii Remote. He now physically switches devices by grabbing the Wii Remote and starts using it for interacting with the video wall system, and by extension, with A. Gestures he makes with the Wiimote are in some way being translated and emulated to inputs for either the keyboard or the mouse.
Whenever he is done working with A, he can switch back to a mouse and keyboard to disable the Wiimote it and regain full control over the system.

With the above scenario, it is easier to understand the reasoning behind the abstractions that were presented in the previous section (Section 4.1.1).

- The first abstraction in the list is more of a requirement for establishing a sound environment for the framework to run in.
- The second is accounted for by looking at the scenario. A video wall application designed for the framework does not *require* the framework to run while the app is running: it can be controlled entirely and separately by the mouse and keyboard. However, whenever the framework is run, the app can consequently be controlled by any device that the framework supports. When switching to such a device, the framework is programmed to map some of the device's inputs to a certain set of mouse and keyboard inputs. This mapping can be compared to an actual mathematical function: the domain would be analogous to device-specific inputs or gestures, and the mapping would map these functions to a certain fixed codomain, which is analogous to a set of inputs for a mouse and keyboard.
- The third abstraction should prevent an external application that makes use of the framework to start tinkering around with devices other than the mouse and keyboard for controlling their UI. This is a job that is meant for the framework, and should not be implemented by an application built upon it.

Mapping methods

The framework relies on a certain way of mapping inputs or gestures of an input device to particular actions on the mouse or the keyboard. The inputs (the domains) for this mapping are device-specific – for example, the Kinect 2 will emulate, say, a left mouse click in response to a different gesture than the Wii Remote will; however, the effect of the mapping (the codomain) will be the same – both devices emulate a left mouse click.

For simplicity, the codomain is presented first. It would be ineffective to create a mapping to *all* of the mouse buttons and keyboard keys combined. This will produce an extremely large codomain, which will make the mapping unnecessarily complex. Instead, the capabilities of the three input devices are used as guidelines for finding a suitable mapping to actions on the mouse and keyboard.

Both the Kinect 2 and the Wii Remote are game controllers. They were designed not only for controlling video games with their motion technology, but also for controlling the interfaces of their respective video game consoles. As will be shown in Section 4.1.3, the GUIs of these consoles are structurally very similar. As a consequence, the set of functions that the Kinect 2 and Wiimote were given to control their respective system's main menu are almost equivalent. Both were meant natively for the following functions: providing a mechanism for following a pointer on screen, clicking/selecting, dragscrolling, zooming in, zooming out, and going to the main menu. These are all pretty common actions in UIs, and for these reasons, they are chosen to emulate the mouse and keyboard in the following way:

- Moving the Wii Remote pointer or Kinect hand cursor will move the mouse cursor;
- Making a select or click gesture will emulate a left mouse click;
- A dragscroll gesture will emulate holding the left mouse button and moving the mouse cursor;
- Zooming in will emulate a + keypress;
- Zooming out will emulate a - keypress;
- Going home will emulate a 0 (zero) keypress.

Next, a set of input gestures – the domain – is determined for each of the three devices. Each of the gestures will emulate one of the actions in the list above:

- Microsoft has built gesture recognition into their Xbox One software for all of the actions in the above list [Microsoft, 2014b]. It is therefore chosen to create a gesture set that is mostly the same as the set that Microsoft is using. This strategy saves the time and effort of creating a custom gesture set[1]. The gestures are displayed in Figures 4.1 to 4.7. All of these images were taken from a gesture tutorial video and have consequently been edited [Microsoft, 2014b]. Some of those gestures are meant for actions that can be considered more real, other are more abstract. Zooming in, for instance, allows for a pretty

[1]Finding the *right* gesture set is outside of the scope of this project. The aim of this research is to find a set with which a usable application framework for interactions with video walls can be implemented.

realistic experience of actually pulling the screen towards you. Clicking is rather more abstract. In this way, the notion of Reality-Based Interaction [Jacob et al., 2008] is accounted for.

- The Leap Motion controller operates by detecting the same gestures as the Kinect 2, with the exception of the ″select″ gesture in Figure 4.3. Instead of pressing towards the screen with a hand, a pointing gesture with the index finger is used, because it was better recognized by the Leap during development of the framework.
- The Wii Remote has a gesture set that differs from the Kinect 2 and the Leap [Nintendo, 2011]. Its functionality relies mostly on buttons and an infrared camera that can make a cursor appear on screen (Figure 3.3). In Nintendo's Wii software, the A button is often used as an equivalent of the left mouse button; in the framework, it can be clicked to select, or held down to dragscroll. Pointing is achieved by using the sensor bar as a reference point for the Wiimote's pointer position in 3D space, and it is used to emulate the mouse's cursor position. The physical plus and minus buttons emulate the functionality for zooming in and out, respectively. The home button emulates a 0 key press on the keyboard.

4.1.3 User interface design

The application's UI is shown in Figure 4.8. The framework's main functions are presented to the user in this dialog box. They are grouped, include enabling HIDs that are supported by the video wall, and allow for tweaking some settings such as the video wall's dimensions and the Wii Remote's sensor bar position. The often used OK, Cancel and Apply buttons, along with the relatively simple and straightforward design, are meant to satisfy the usability measurement of learnability and memorability [Nielsen, 1994].
The app is designed as a Windows task tray application. When programs like these are closed or minimized, they reside in the bottom right of the task bar, until they are manually quit. This allows for apps to keep running in the background, which is ideal for continuously tracking gestures in this case.

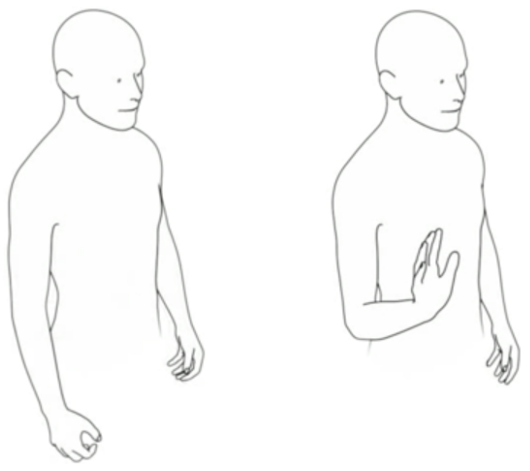


Figure 4.1: Raising a hand starts updating the mouse cursor to the hand's position.

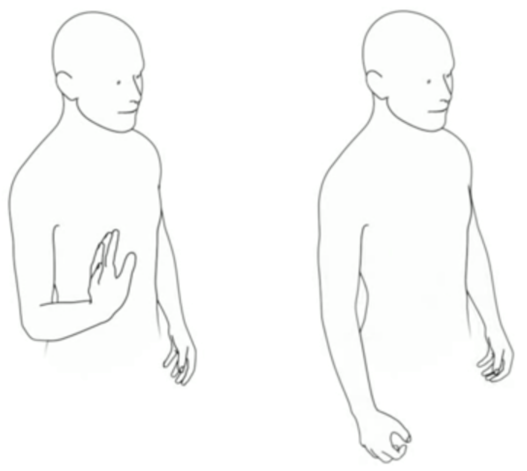


Figure 4.2: Lowering a hand stops updating the mouse cursor to the hand's position.

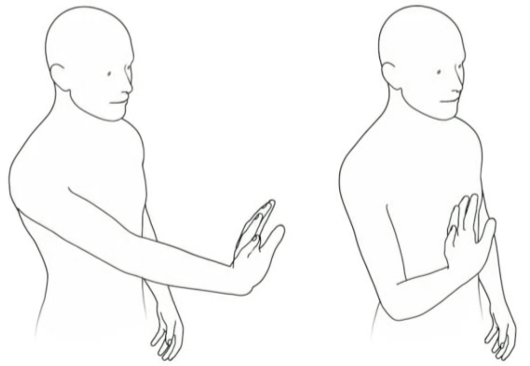


Figure 4.3: Moving a hand towards the screen and back selects an item by emulating a left mouse click.

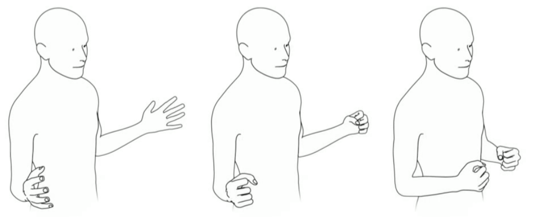


Figure 4.4: You can go home by grabbing the sides of the screen and pulling inwards. This emulates a 0 keypress.

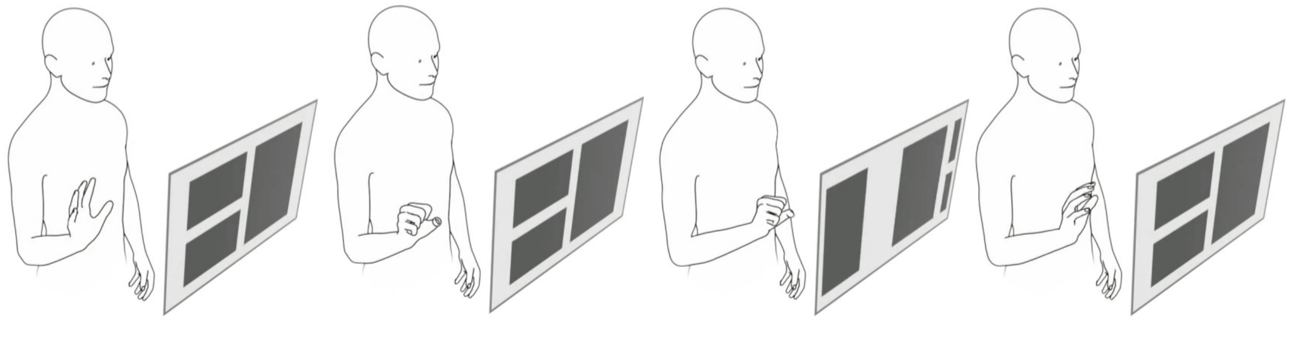


Figure 4.5: Grabbing the screen and moving it around is equivalent to left clicking and dragging the mouse around.

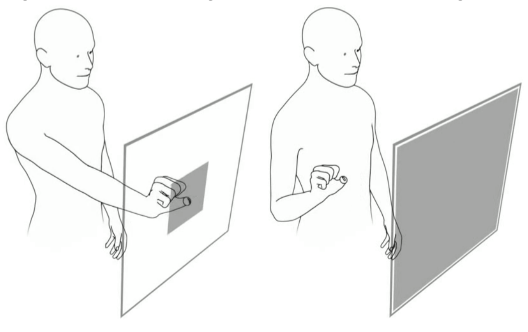


Figure 4.6: Grabbing the screen and pulling it towards you zooms in by emulating a + keypress.

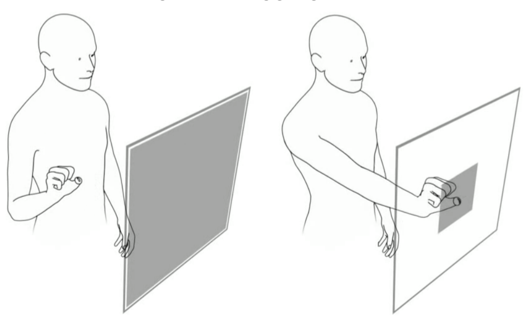
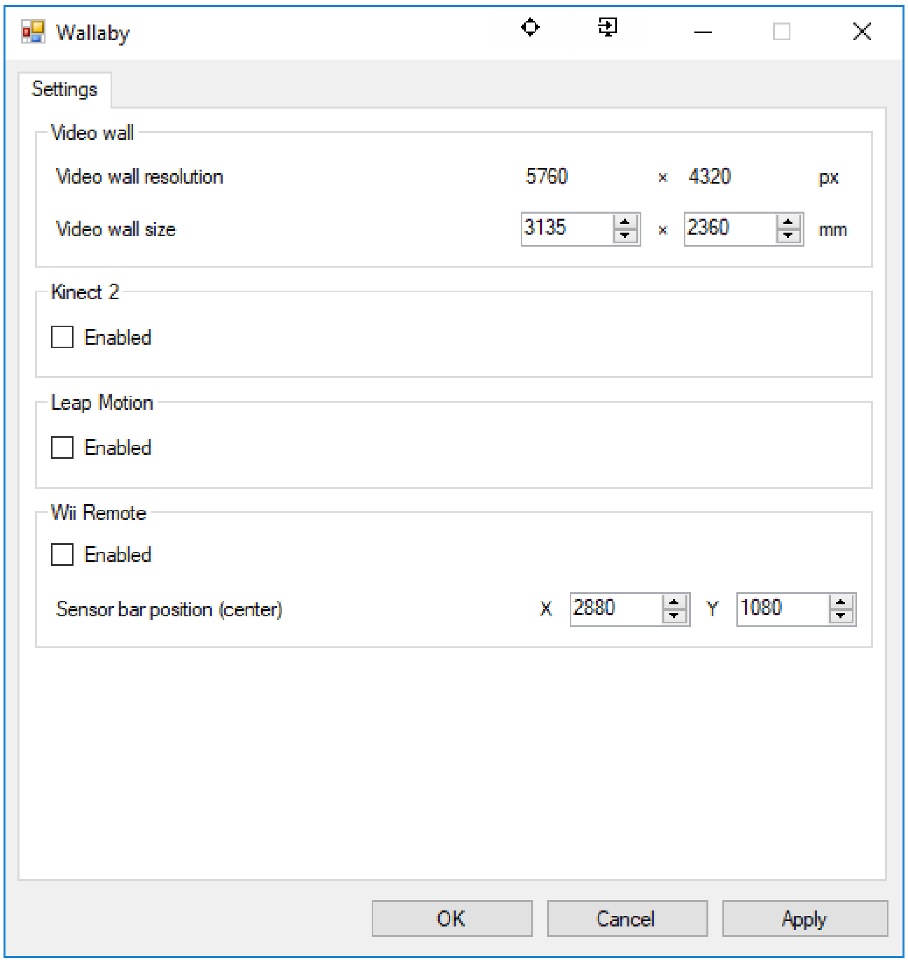


Figure 4.7: Grabbing the screen and moving it away from you zooms out by emulating a - keypress.


Wallaby

Settings

Video wall

Video wall resolution 5760 × 4320 px

Video wall size 3135 × 2360 mm

Kinect 2

☐ Enabled

Leap Motion

☐ Enabled

Wii Remote

☐ Enabled

Sensor bar position (center) X 2880 Y 1080

Figure 4.8: The UI of the application framework.

4.2 A high-resolution image gallery for video walls

4.2.1 Core features

The gallery for high-resolution imagery is designed as an arbitrary application on which the application framework can be tested. As mentioned in Chapter 4, the framework is designed to ensure support for human interface devices that emulate the following set of mouse and keyboard actions in a user interface:

- Moving the mouse;
- Clicking the left mouse button;
- Holding the left mouse button down and dragging the cursor;
- Zooming in and out pressing the + and - keys on the keyboard;
- Going ”home” by pressing the 0 key on the keyboard.

According to the framework's standards, the gallery application should be designed so that most of it is controllable by this limited set of actions.

Creating a design for the gallery app starts by organizing the high-resolution imagery. It will be run on a video wall, a big screen, and the app can be controlled by several devices. The app is a gallery, which means it should show a collection of images the user can navigate through and eventually make a selection out of. When viewing an image, the user should initially be shown the image entirely. Because the images are so large, the user should be able to zoom in and out, to navigate parts of the image, and to return to the default zoom level. Additionally, the user should be able to view some information about the image.

4.2.2 User interface design

Thinking about the app in terms of how the user will utilize it allows for splitting the user interface into multiple navigational parts. The initial flow of navigation starts with a collection view of images that the gallery has to offer. There might exist multiple collections that the user should be able to choose from, so the flow could continue from here to another collection view. However, if the user selects one of the images, the interface must provide some form of navigation from the collection view to the image view.

Let's first consider the gallery's collection views. These are designed according to familiar user interface characteristics that are present in the Xbox One's and Wii's main interfaces – the native interfaces that were designed for interactions with the Kinect and the Wii Remote. Thus, let's take a look at the Wii's and the Xbox One's home menus. These are shown in Figures 4.9 and 4.10.

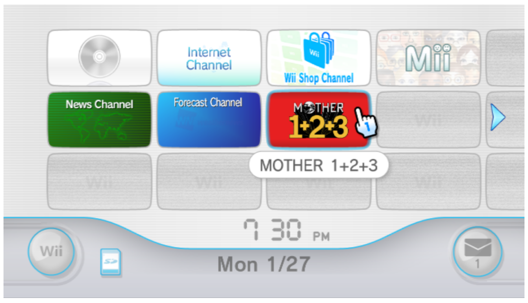


Figure 4.9: The Wii Menu.

Source: [Starmen.Net, 2013]

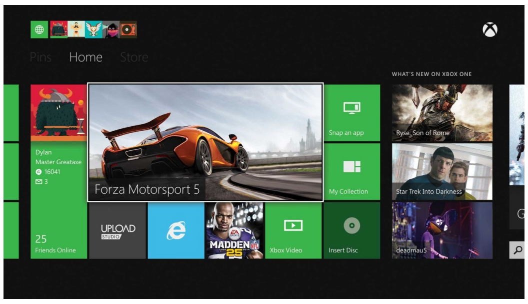


Figure 4.10: The Xbox One Dashboard.

Source: [TechnoBuffalo LLC, 2015]

While being entirely different systems and interaction devices, their user interfaces show great structural similarities. These are shown in Figures 4.11 and 4.12.

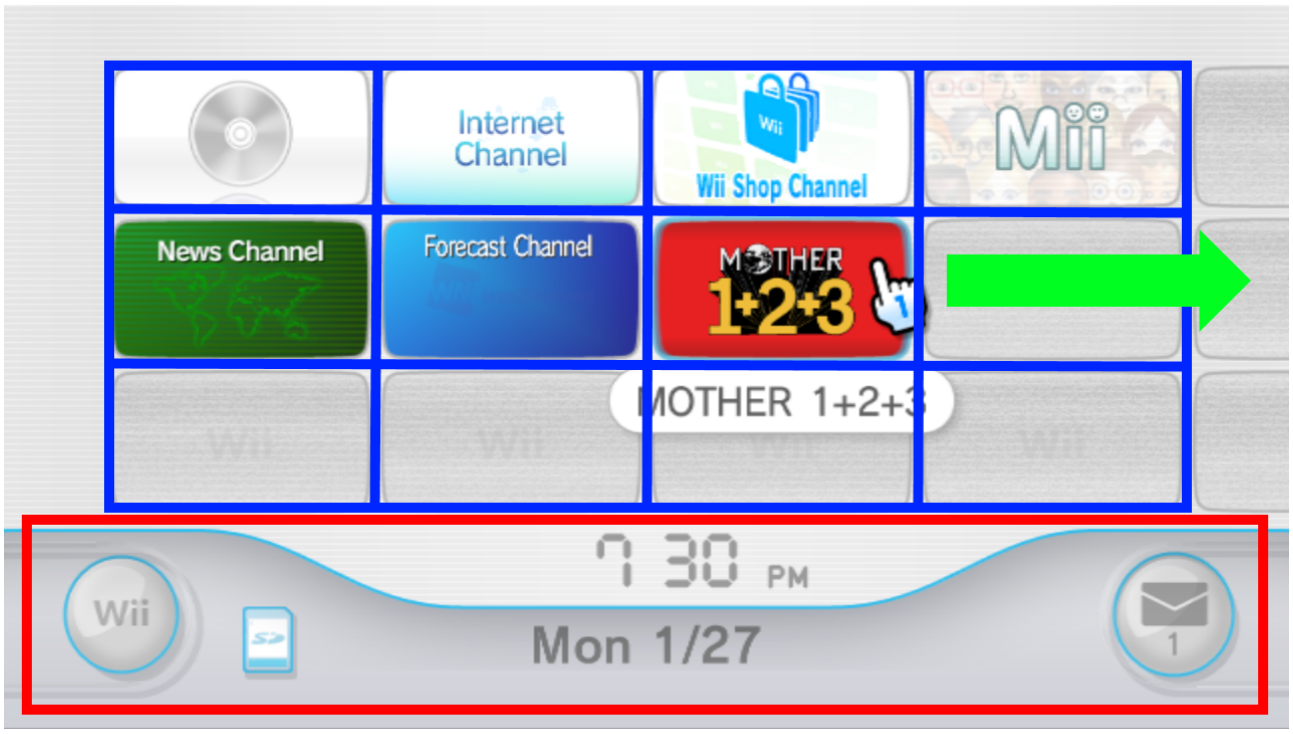


Figure 4.11: The Wii Menu's design, in which particular UI structures are highlighted.

Adapted from: [Starmen.Net, 2013]

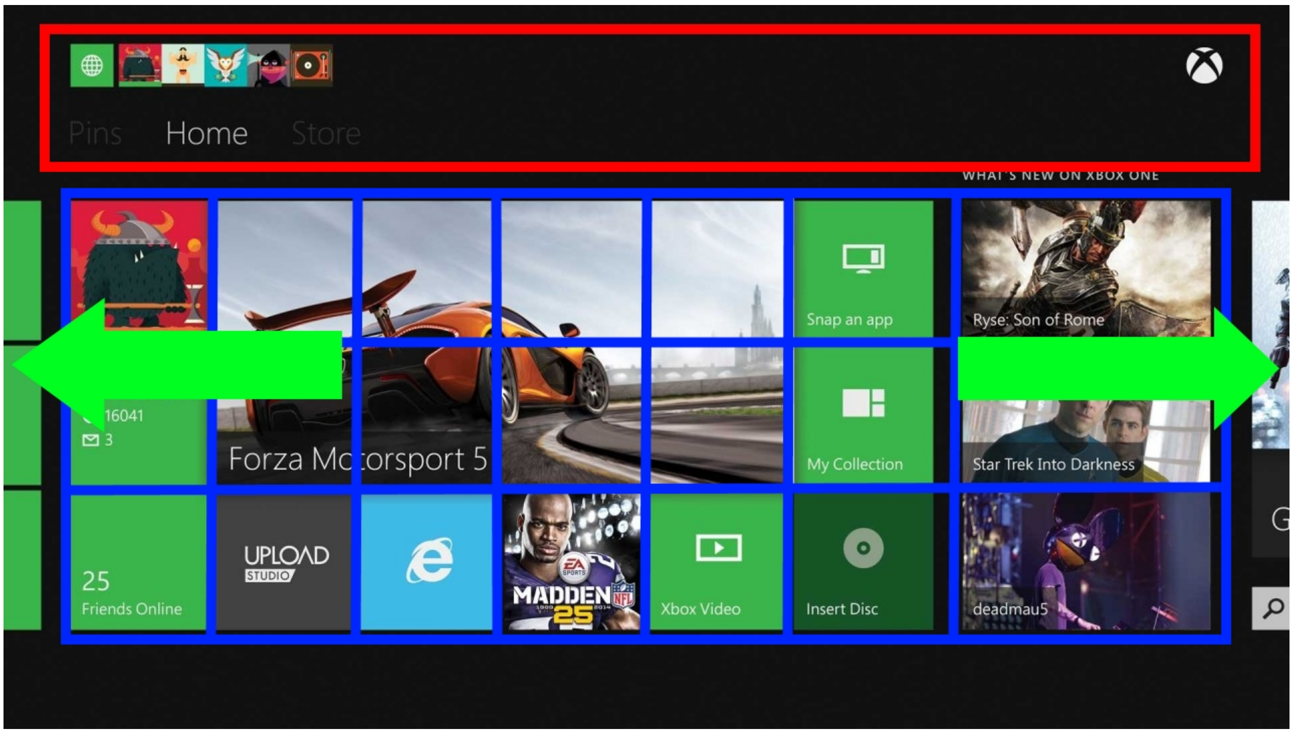


Figure 4.12: The Xbox One Dashboard's design, in which particular UI structures are highlighted.

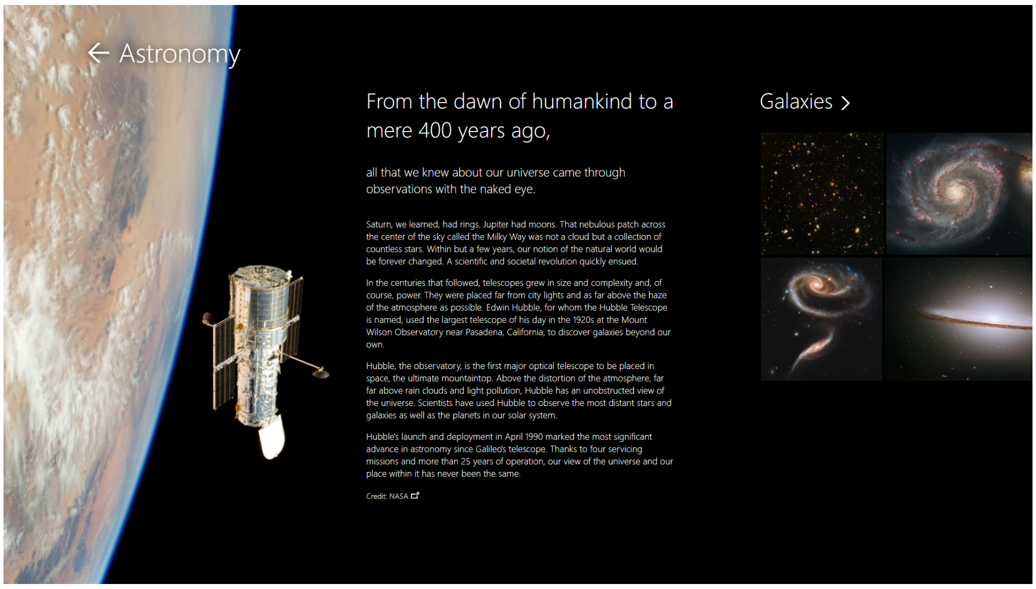
Adapted from: [TechnoBuffalo LLC, 2015]

Both interfaces present a collection of apps in their home menus, which are highlighted in blue, and that make up most of the UI as they provide their system's main functionality. The collection view has the form of a two-dimensional grid in which each cell belongs to a specific app. The cells themselves are rather large, something that has probably been implemented for learnability and to increase the margin of error for users attempting to select the cells with a motion-based device [Nielsen, 1994].

Secondary functionality is available through a screen-wide menu bar, highlighted in red. These provide for accessing system features apart from than apps.

The last thing of note is that both interfaces have specifically chosen horizontal over vertical flow of navigation. Users can navigate to collections with even more apps by scrolling left and right. Vertical orientation is sporadically used to break the flow, as is visible in the bottom left grid cell of the Xbox Dashboard.

Based on these interfaces, the following design has been created for the collection views of the gallery application (Figure 4.13).



From the dawn of humankind to a mere 400 years ago,

all that we knew about our universe came through observations with the naked eye.

Saturn, we learned, had rings. Jupiter had moons. That nebulous patch across the center of the sky called the Milky Way was not a cloud but a collection of countless stars. Within but a few years, our notion of the natural world would be forever changed. A scientific and societal revolution quickly ensued.

In the centuries that followed, telescopes grew in size and complexity and, of course, power. They were placed far from city lights and as far above the haze of the atmosphere as possible. Edwin Hubble, for whom the Hubble Telescope is named, used the largest telescope of his day in the 1920s at the Mount Wilson Observatory near Pasadena, California, to discover galaxies beyond our own.

Hubble, the observatory, is the first major optical telescope to be placed in space, the ultimate mountaintop. Above the distortion of the atmosphere, far far above rain clouds and light pollution, Hubble has an unobstructed view of the universe. Scientists have used Hubble to observe the most distant stars and galaxies as well as the planets in our solar system.

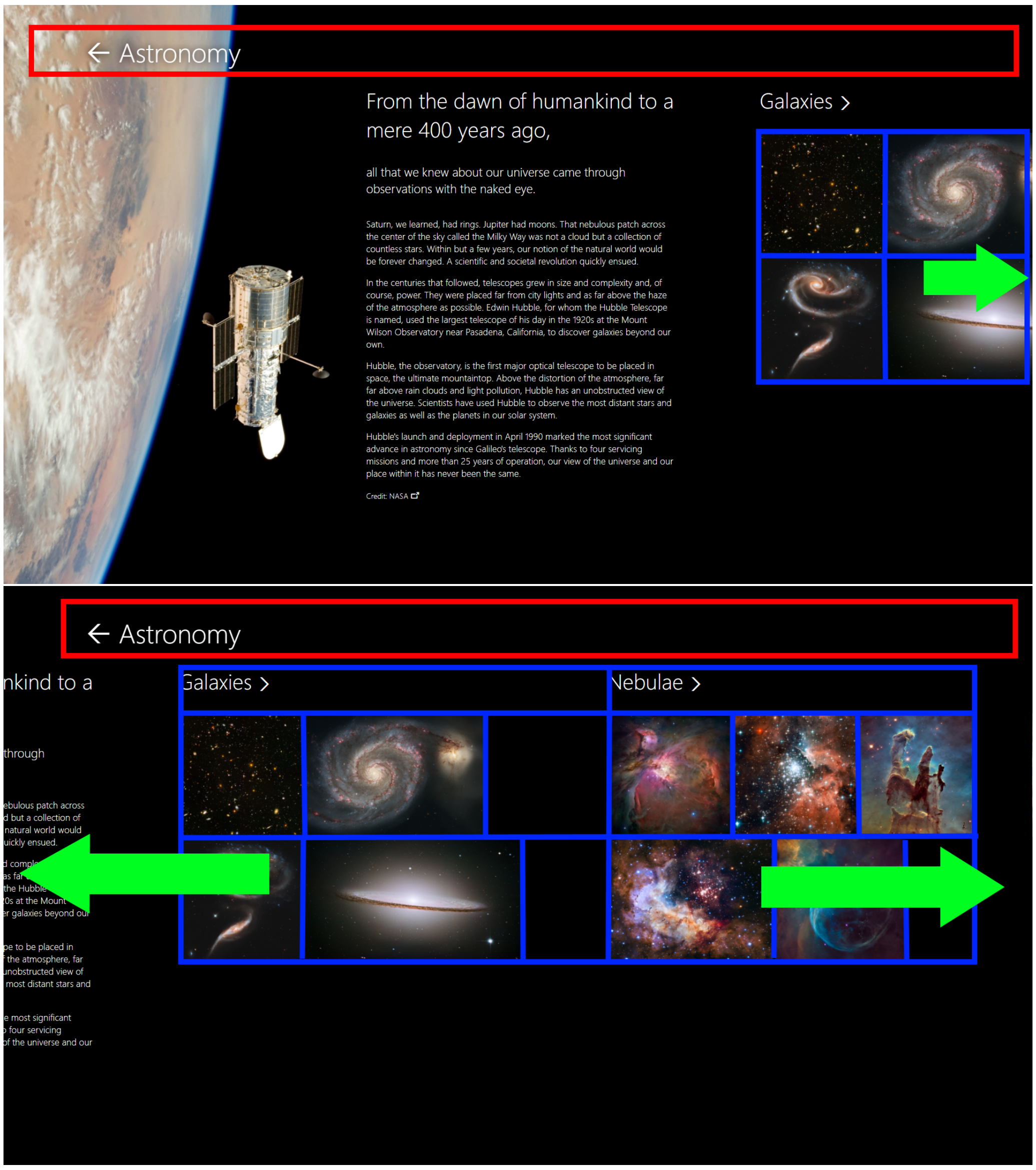
Hubble's launch and deployment in April 1990 marked the most significant advance in astronomy since Galileo's telescope. Thanks to four servicing missions and more than 25 years of operation, our view of the universe and our place within it has never been the same.

Credit: NASA

Figure 4.13: The UI of a collection view of the gallery application.

For clarity, the image is included again (Figure 4.14), but this time, particular structures of the UI are highlighted. In this example, a collection view featuring high-resolution Hubble telescope images is brought into view. Collections of grouped images are highlighted in blue. They are placed at the right, as to indicate end users that the flow of navigation is horizontal. Users can make the cursor – which is replaced by a large hand icon – appear on screen and dragscroll the interface to bring the collections into view. The collection cells are containers for clickable thumbnails of the high-resolution gallery images. Additionally, collections can be nested within the interface of parent collections, i.e. the collection of Astronomy images has two subcollections, called Galaxies and Nebulae. When a user selects their titles, the respective collection view is brought into view, which has a layout similar to this one. The red bar on top provides a means of navigating back to the previous collection view, and the green arrows indicate the flow of control. Each collection is attributed with its own image and explanatory notes. The vertical orientation of the notes break the horizontal flow and make the entire collection visually appealing.

Next, the gallery should provide an image view. This should feature the image entirely, while still providing for the functions the user expects, such as zooming. The interface of the image view is shown in Figure 4.15.



From the dawn of humankind to a mere 400 years ago,

all that we knew about our universe came through observations with the naked eye.

Saturn, we learned, had rings. Jupiter had moons. That nebulous patch across the center of the sky called the Milky Way was not a cloud but a collection of countless stars. Within but a few years, our notion of the natural world would be forever changed. A scientific and societal revolution quickly ensued.

In the centuries that followed, telescopes grew in size and complexity and, of course, power. They were placed far from city lights and as far above the haze of the atmosphere as possible. Edwin Hubble, for whom the Hubble Telescope is named, used the largest telescope of his day in the 1920s at the Mount Wilson Observatory near Pasadena, California, to discover galaxies beyond our own.

Hubble, the observatory, is the first major optical telescope to be placed in space, the ultimate mountaintop. Above the distortion of the atmosphere, far far above rain clouds and light pollution, Hubble has an unobstructed view of the universe. Scientists have used Hubble to observe the most distant stars and galaxies as well as the planets in our solar system.

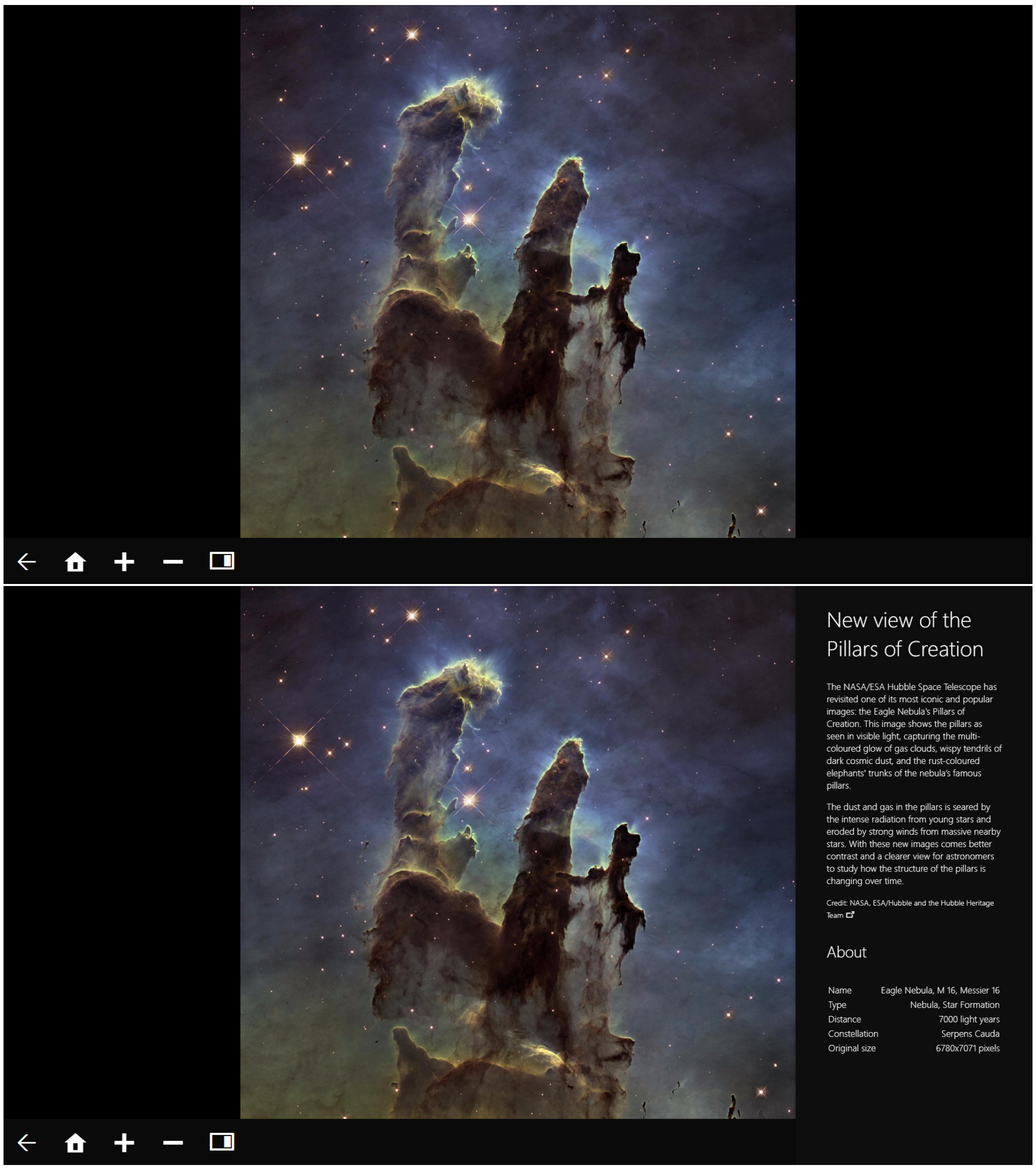
Hubble's launch and deployment in April 1990 marked the most significant advance in astronomy since Galileo's telescope. Thanks to four servicing missions and more than 25 years of operation, our view of the universe and our place within it has never been the same.

Credit: NASA

Figure 4.14: The UI of a collection view of the gallery application in which certain structures are highlighted.

The interface consists of the image and a menu bar with several buttons that control (from left to right) going back to the parent collection view, setting the zoom level to the default value, zooming in, zooming out, and showing or hiding a sidebar containing some image metadata. If the sidebar content were to extend beyond the view, dragscrolling is enabled to scroll it into view.
The user can control the image by other means than only the buttons in the menu bar. This is where the device-specific gestures come into play. Moving the cursor makes a hand appear on screen. Clicking a certain location in the image zooms in, and the image can be dragged around by using the dragscroll gesture. Zooming in and out is also possible through gestures, as is going home, which resets the zoom level.

Having a menu bar besides having gestures provides users with two ways to achieve the same goals, which should make it somewhat easier for them to accomplish basic tasks and improve learnability. Having alternatives can also make their errors decrease and benefits memorability. This improves usability and may prove beneficial in during the user tests.



New view of the Pillars of Creation

The NASA/ESA Hubble Space Telescope has revisited one of its most iconic and popular images: the Eagle Nebula's Pillars of Creation. This image shows the pillars as seen in visible light, capturing the multi-coloured glow of gas clouds, wispy tendrils of dark cosmic dust, and the rust-coloured elephants' trunks of the nebula's famous pillars.

The dust and gas in the pillars is seared by the intense radiation from young stars and eroded by strong winds from massive nearby stars. With these new images comes better contrast and a clearer view for astronomers to study how the structure of the pillars is changing over time.

Credit: NASA, ESA/Hubble and the Hubble Heritage Team

About

Name	Eagle Nebula, M 16, Messier 16
Type	Nebula, Star Formation
Distance	7000 light years
Constellation	Serpens Cauda
Original size	6780x7071 pixels

Figure 4.15: The UI of an image view of the gallery application. The first image shows the initial interface; the second show the interface after the sidebar button has been clicked.

Chapter 5

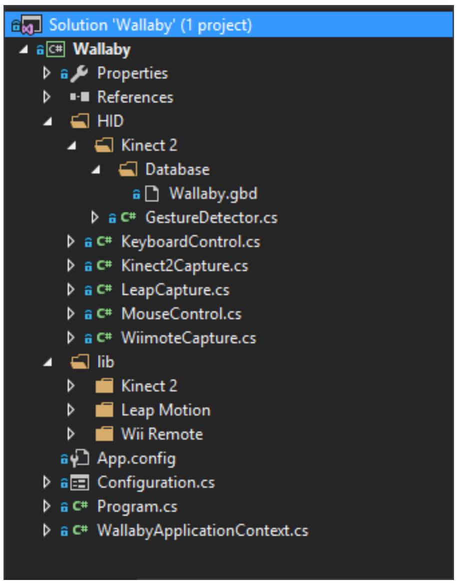
Implementation

5.1 Implementation of the application framework

The framework design introduced in Chapter 4 is implemented by creating a so-called Windows Forms Application in Visual C#. These applications are meant to provide end users with a simple GUI to control the application's features and settings. The design of the interface is set out in section 4.2, and the interface is shown in Figure 4.8. In the UI, users are presented with options for enabling the HIDs that are currently supported by the framework – that is, the Kinect 2, the Leap Motion controller, and the Wii Remote controller. They can also set some variables that slightly alter the way the framework behaves based on the specifications of the video wall they're using. These include the video wall's physical dimensions and the position of the sensor bar that is used for interactions with the Wii Remote.

Internally, the values regarding the video wall's dimensions and the sensor bar are used to the determine a reference window in which the cursor should move, based on a user's hand or the Wiimote position. The checkboxes activate or stop the connections that the framework establishes with the interface devices.

In the Visual Studio project, the libraries mentioned in Section 3.3 are included in a separate folder for each of the HIDs. Another folder contains the logic for capturing the Kinect 2, the Leap, and the Wiimote, and for emulating the mouse and keyboard. Each device is assigned a file in which it's class resides. If the framework proves successful, it will be made open source, so that anyone can contribute by improving it or adding support for other devices by including their own class files.



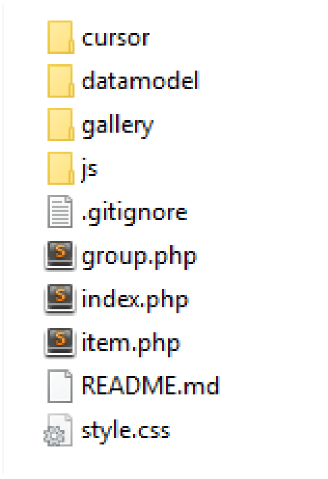
```
Solution 'Wallaby' (1 project)
  Wallaby
    Properties
    References
    HID
      Kinect 2
        Database
          Wallaby.gbd
        GestureDetector.cs
      KeyboardControl.cs
      Kinect2Capture.cs
      LeapCapture.cs
      MouseControl.cs
      WiimoteCapture.cs
    lib
      Kinect 2
      Leap Motion
      Wii Remote
    App.config
    Configuration.cs
    Program.cs
    WallabyApplicationContext.cs
```

Figure 5.1: The internal structure of the framework, which is named Wallaby.

The structure can be viewed in Figure 5.1. Support for more devices can be added by adding a separate class file to /HID folder. Device libraries can be added in the /lib folder.
The /HID folder contains another folder for maintaining a gesture database file for the Kinect 2. Gestures are detected by the Kinect library at runtime by providing it with this gesture database file. A gesture database is created by first capturing videos of all the available gestures with Kinect Studio. Next, the video files can be imported into a program called Visual Gesture Builder. Both of these programs are part of Microsoft's Kinect for Windows v2 SDK [Microsoft, 2014a]. With Visual Gesture Builder, the specific gestures are then tagged wnenever they occurred. These tagged video files serve as training data for the final step, in which a machine learning algorithm is activated that generates a Visual Gesture Builder database for detecting gestures at run-time.

5.2 Implementation of the gallery application

The gallery application, called Magnifeye, is written in PHP, HTML, CSS, and Javascript, and is run through a local server with XAMPP [Apache Friends, 2017]. The folder structure is shown in Figure 5.2. It consists of two main PHP files: one template for the collection view, `group.php`, and one template for the image view, `item.php` (section 4.2.2). The `gallery` directory contains a hierarchical folder structure matching the structure of all the nested collections and image files in the gallery. In this structure resides the collections' featured image files and the gallery's high-resolution image files. The `datamodel` folder provides a model for containing and querying the hierarchical metadata (XML) for the collections and the images, such as the sidebar data for each image.


```
cursor
datamodel
gallery
js
.gitignore
group.php
index.php
item.php
README.md
style.css
```
Figure 5.2: The internal structure of the gallery application, which is named Magnifeye.

When starting the gallery application, the `group.php` file is invoked without any arguments. The PHP script then attempts to load the root collection by searching the datamodel folder for XML metadata of the root collection. When it is found, the appropriate XML node is returned and broken down into chunks that are used to fill the collection view template with the correct collection data. In case an argument for locating a specific collection was given, it is looked up in the XML feed by checking the descendants of the root collection recursively.

The `item.php` file is a template that relies on the same principle for the most part: it looks up image metadata based on the argument it was given and loads the image found into a third-party image viewer.

Because the high-resolution images are rather large – often larger than 5000×5000 pixels – loading them into memory entirely causes significant overhead. To avoid this, each image is split into multiple smaller tiles, in the way illustrated in Figure 5.3. An optimized imaging tool called VIPS [Cupitt, 2017] is used to this end.

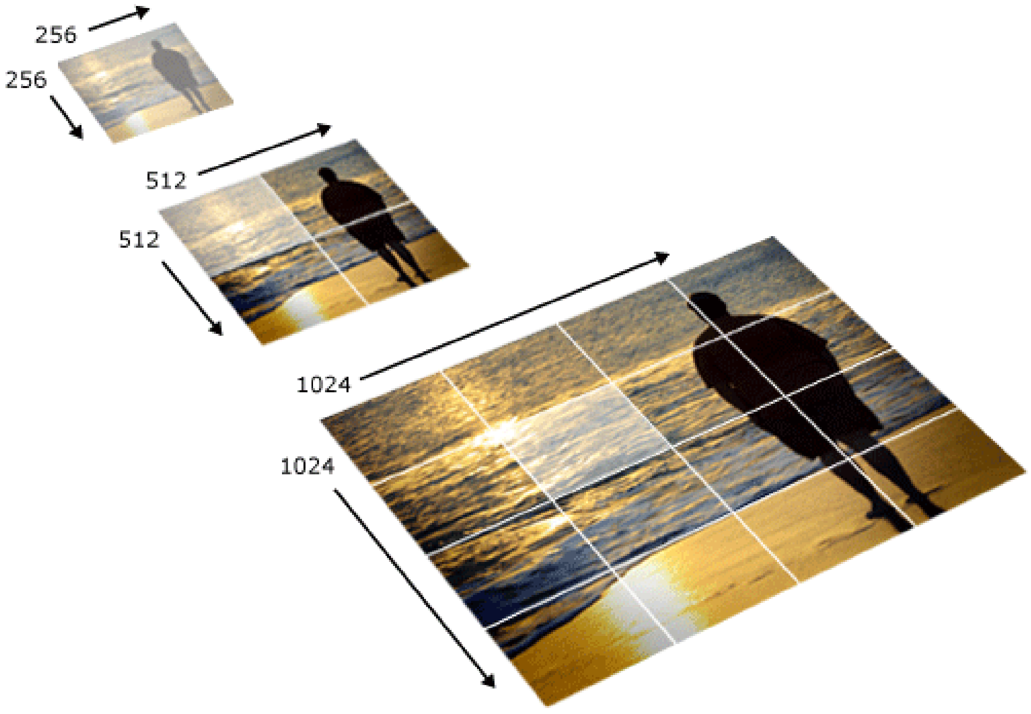


Figure 5.3: An example of a deep-zoom image file. When viewing a large image while zoomed in on the highlighted area, only that area is loaded into memory instead of the entire 1024 × 1024 image.

Source: [Microsoft, 2017]

The serving of these tiles require an optimized solution so that the video wall's performance is not affected, even when running the gallery application in full-screen. For this end, the OpenSeadragon viewer is utilized [CodePlex Foundation and OpenSeadragon contributors, 2017].

Chapter 6

Experiments

6.1 User tests and testing procedure

The experiment in this chapter is meant to gauge to what extent the generic application framework for video walls conforms to the measurements of usability [Nielsen, 1994]. To test the framework, a dozen users have been subjected to the following test protocol.

Each participant is introduced to the concept of a generic application framework for interactions with video walls. It is emphasized that they are about to take part in a test of an application that will allow other video wall applications to make use of the interactions provided by the framework. They are shown the interaction devices they are about to work with and confirm that, after this introduction, they have a basic understanding of how it all works.

Next, they were briefly shown the arbitrary video wall application – that is, the image gallery – with which they shall shortly interact by using the framework's functionality. They were also explained that the framework itself is a Windows application that runs in the OS's application tray.

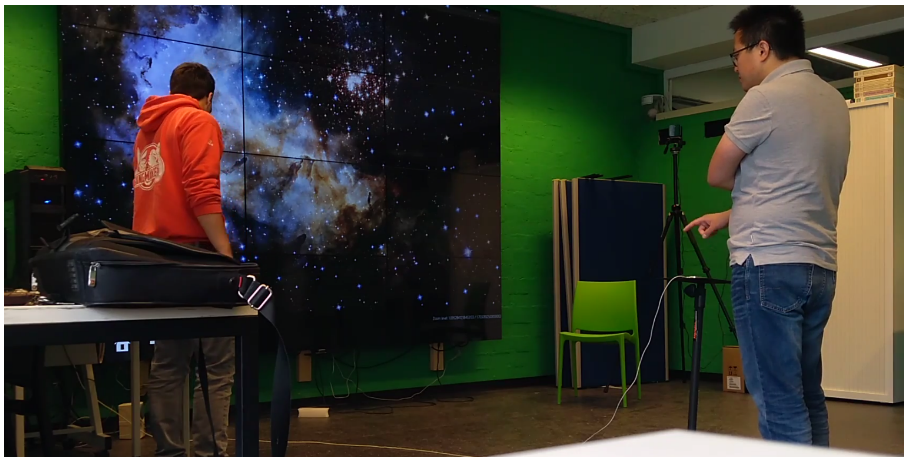


Figure 6.1: Image taken during initialization with the Leap Motion controller.

Consequently, they were asked to follow verbal instructions according to the next usability specification for performing specific tasks in the UI of the gallery application by using the Leap Motion controller and the Wii Remote controller.[1][2]

Time	Task	Measured	Current	Worst	Planned	Best
Initialization	Start the application framework, activate Leap Motion, make hand appear on screen	Time to pointer tracking		60 seconds	20 seconds	8 seconds
Initial performance #1	Perform each supported gesture/action once with Leap	Time to complete		300 seconds	150 seconds	80 seconds
Navigation #1	Navigation task that includes supported actions/gestures at least once with Leap	# errors		20	8	3
Navigation #1	Navigation task that includes supported actions/gestures at least once with Leap	Time to complete		230 seconds	150 seconds	60 seconds
Switch devices	Go back to application, disable Leap, enable Wii Remote, make hand appear on screen	Time to complete		40 seconds	20 seconds	5 seconds
Initial performance #2	Perform each supported gesture/action once with Wiimote	Time to complete		80 seconds	60 seconds	20 seconds
Navigation #2	Navigation task that includes supported actions/gestures at least once	# errors		10	5	2
Navigation #2	Navigation task that includes supported actions/gestures at least once	Time to complete		200 seconds	100 seconds	40 seconds
First impression	SUS Questionnaire	Likert scale score		68	75	100

At the end of the test, the participants are asked some questions specific to their iteration through the testing procedure, and they are asked to fill in the System Usability Scale questionnaire (Appendix A). This questionnaire is a quick and dirty tool for measuring usability of a certain system. The participants were asked to

[1]The Kinect 2 is excluded from the user tests, because it malfunctions on the video wall setup that the tests were performed on as well.

[2]The home gesture for the Leap was not implemented for this test due to poor gesture detection performance during development.

specifically rate the application framework in accordance with the gallery application, and to not pass their judgment just for one of these two components.

6.2 Results

The following results were obtained from the testing procedure in the previous section. The values in the Measured column represent the mean scores.

Time	Task	Measured	Current	Worst	Planned	Best
Initialization	Start the application framework, activate Leap Motion, make hand appear on screen	Time to pointer tracking	64 seconds	60 seconds	20 seconds	8 seconds
Initial performance #1	Perform each supported gesture/action once with Leap	Time to complete	249 seconds	300 seconds	150 seconds	80 seconds
Navigation #1	Navigation task that includes supported actions/gestures at least once with Leap	# errors	9.6	20	8	3
Navigation #1	Navigation task that includes supported actions/gestures at least once with Leap	Time to complete	186 seconds	230 seconds	150 seconds	60 seconds
Switch devices	Go back to application, disable Leap, enable Wii Remote, make hand appear on screen	Time to complete	60 seconds	40 seconds	20 seconds	5 seconds
Initial performance #2	Perform each supported gesture/action once with Wiimote	Time to complete	83 seconds	80 seconds	60 seconds	20 seconds
Navigation #2	Navigation task that includes supported actions/gestures at least once	# errors	4.3	10	5	2
Navigation #2	Navigation task that includes supported actions/gestures at least once	Time to complete	243 seconds	200 seconds	100 seconds	40 seconds
First impression	SUS Questionnaire	Likert scale score	75.41	68	75	100

Chapter 7

Conclusions & Discussion

Based on the results of the testing procedure, the following can be said conclusively on generating a framework for interactions with video walls.

Let's first look at the results from the usability specification. The time people need for initialization or task switching, tasks that are performed in the tray application, was fairly underestimated. No one was able to start the application and enable the Leap Motion controller initially within the planned time frame of 20 seconds. It seemed to take them rather long to interpret the interface of the application framework. The average (64) here may be somewhat misleading, because values alternate all the way from 21 to 102 seconds. This is also the case for the task of switching devices. The mean time frame of 60 seconds includes values that jump from 15 to 130 seconds, and only one was able to do it in under 20 seconds (15). This indicates that the learnability aspect of the framework app can still be improved.

On to the performance tasks with the Leap Motion controller. Users overall experienced some problems with the precision of the gesture detection for some of the gestures. In general, moving the cursor with the hand and dragscrolling went very well; clicking a specific item on screen proved hard, but learnable; and zooming in or out was regarded by most as being too inconsistent. These problematic gestures were the cause of most of the user's errors (9.6) with the Leap. Some participants indicated that they thought they were to blame for not being able to successfully zoom in or out by using gestures, while they were actually not. Admittedly, the recognition for zooming gestures was programmed in a very strict way to ensure consistency, but this lack of flexibility can apparently be a downfall as well.

The gallery application's UI appeared to suit the gesture set of the Leap pretty well. No one really hesitated in trying to navigate their way around, which is an indication of a learnable and efficient UI. Participants did sometimes have a hard time in going back from an image view to a collection view, because the button that needs to be clicked for going back is all the way in the bottom left corner of the video wall display. This meant having to sometimes make a little awkward pointing gesture just on the edge of the Leap's hand detection range. Nobody really got discouraged by this, though.

It can be said that users were pretty satisfied with the way the gallery was functioning. Some did complain about performance issues that arose due to this specific video wall setup, as it made them feel like they were doing something wrong.

Switching devices to the Wiimote went almost without trouble for anybody. There were some problems with accidental clicks in certain scenarios, which was due to programming, but people unanimously felt the Wiimote was easier to use compared to the Leap. The number of errors in similar navigational tasks halved, but this might also be caused by the participants becoming used to the system. Either way, it's a plus.

Despite the Leap's gesture recognition being error-prone, it was regarded as the device that was "more fun" to use than the Wiimote, in retrospect. This might be due to the notion of Reality-Based Interaction [Jacob et al., 2008]. It does seem to come into play when interacting in a rather reality-based way, which is the case for the Leap more than it is for the Wiimote. That might be due to the tradeoff between the efficiency of having some abstract and unrealistic elements – such as the buttons and image miniatures – and the expressive power of some real ones – such as the realistic zooming gestures and the pointer control.

Finally, the results of the System Usability Scale. More than half of the participants had trouble understanding the concept of the application framework initially. They had to be made explicitly clear before and after the testing procedure (but before the questionnaire) that the test they participated in, was not meant for just the gallery application. In the end, only 1 out of 12 was not sure whether he found the system unnecessarily complex; the others either disagreed (2) or strongly disagreed (9).

The lion's share of the participants praise the ease of use of the entire system, and did generally not feel like they or others would have to learn a lot before being able to get going this system. However, the planned time levels in the system usability scale imply otherwise. Almost all of these have estimated shorter time frames for the tasks performed. This does not necessarily imply low rates of efficiency and learnability – rather, it implies an overestimation of the capabilities of the test participants.

Most of the SUS ratings are above average, and the system gets an overall SUS score of 75.41. Because these scores are not percentile-based, but on a Likert scale, a conversion method can be used [Sauro, 2011] to estimate that the measurement of usability is around 75%, which is spaciously more than the average of a score of 68, or 50%.

To conclude: yes, an application framework that governs interactions for video walls is shown to be usable. It would be a good effort to encourage open development, so that a framework like this can be improved and applied to many more instances of video walls for those who seek to build interactive applications for them.

Bibliography

[Apache Friends, 2017] Apache Friends (2017). XAMPP for Windows 5.6.31. https://www.apachefriends.org/index.html. Retrieved August, 2017.

[CodePlex Foundation and OpenSeadragon contributors, 2017] CodePlex Foundation and OpenSeadragon contributors (2017). Openseadragon. https://github.com/openseadragon/openseadragon. Retrieved August, 2017.

[Cupitt, 2017] Cupitt, J. (2017). libvips : an image processing library. https://github.com/jcupitt/libvips. Retrieved August, 2017.

[Electronic Industries Association. Engineering Department., 1969] Electronic Industries Association. Engineering Department. (1969). *Interface between data terminal equipment and data communication equipment employing serial binary data interchange.* Electronic Industries Association, Engineering Dept., Washington.

[Gek Siong, 2003] Gek Siong, L. (2003). Video Wall Displays. http://xenon.stanford.edu/~geksiong/papers/cs147assignment.html. Retrieved August, 2017.

[Jacob et al., 2008] Jacob, R. J., Girouard, A., Hirshfield, L. M., Horn, M. S., Shaer, O., Solovey, E. T., and Zigelbaum, J. (2008). Reality-based Interaction: A Framework for post-WIMP Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 201–210, New York, NY, USA. ACM.

[Koninklijke Philips N.V., 2011] Koninklijke Philips N.V. (2011). RS232 Serial Interface Communication Protocol (SICP v1.87). https://shop.kindermann.de/erp/KCO/avs/3/3007/3007000000/01_Anleitungen+Doku/RS232Codes.pdf. Retrieved August, 2017.

[Langa, 2017] Langa, L. (2017). Configparser 3.5.0. https://bitbucket.org/ambv/configparser. Retrieved August, 2017.

[Leap Motion, Inc., 2016] Leap Motion, Inc. (2016). Leap Motion Orion 3.2.0. https://developer.leapmotion.com/releases/?category=orion. Retrieved August, 2017.

[Liechti, 2017] Liechti, C. (2017). pySerial 3.4. https://github.com/pyserial/pyserial/. Retrieved August, 2017.

[Lumo Interactive, 2016] Lumo Interactive (2016). https://static1.squarespace.com/static/55b7aa78e4b010ebd421bb07/567e0bea0e4c117448819fc4/56c543b9d51cd4ce64d29d94/1455916653328/?format=1500w. Retrieved August, 2017.

[Lundh, 1999] Lundh, F. (1999). Tkinter. https://users.tricity.wsu.edu/~bobl/cpts481/an-introduction-to-tkinter.pdf. Retrieved August, 2017.

[Malmgren, 2017] Malmgren, A. (2017). FreePIE (Programmable Input Emulator). http://andersmalmgren.github.io/FreePIE/. Retrieved August, 2017.

[MarketsandMarkets.com, 2015] MarketsandMarkets.com (2015). Video Wall Market by Product, Application, Vertical & Geography - Forecast to 2020. http://www.marketsandmarkets.com/Market-Reports/video-wall-market-101439164.html. Retrieved July, 2017.

[Microsoft, 2014a] Microsoft (2014a). Kinect for Windows SDK 2.0. https://www.microsoft.com/en-us/download/details.aspx?id=44561. Retrieved August, 2017.

[Microsoft, 2014b] Microsoft (2014b). Official Kinect 2 Gesture Tutorial. https://www.youtube.com/watch?v=P4naZ58wIto. Retrieved August, 2017.

[Microsoft, 2017] Microsoft (2017). https://img-prod-cms-rt-microsoft-com.akamaized.net/cms/api/am/imageFileData/RE1msKf?ver=f08e&q=90&m=6&h=450&w=800&b=%23FFFFFFFF&l=f&o=t. Retrieved August, 2017.

[Microsoft, 2017] Microsoft (2017). https://i-msdn.sec.s-msft.com/dynimg/IC141135.png. Retrieved August, 2017.

[Microsoft, 2017] Microsoft (2017). Visual Studio IDE, Code Editor, VSTS, & Mobile Center. https://www.visualstudio.com/. Retrieved August, 2017.

[My Nintendo News, 2013] My Nintendo News (2013). https://sickr.files.wordpress.com/2011/05/wii-remote-jacket.jpg. Retrieved August, 2017.

[Nielsen, 1994] Nielsen, J. (1994). *Usability engineering*. Elsevier.

[Nintendo, 2011] Nintendo (2011). *Wii Operations Manual: System Setup*. Retrieved August, 2017.

[Peek, 2009] Peek, B. (2009). Managed Library for Nintendo's Wiimote v1.7.0.0. https://github.com/BrianPeek/WiimoteLib. Retrieved August, 2017.

[Prokashev, 2016] Prokashev, D. (2016). dragscroll. https://github.com/asvd/dragscroll. Retrieved August, 2017.

[Sauro, 2011] Sauro, J. (2011). https://measuringu.com/sus/. Retrieved August, 2017.

[Starmen.Net, 2013] Starmen.Net (2013). https://ssl-forum-files.fobby.net/forum_attachments/0035/5667/menu1.png. Retrieved August, 2017.

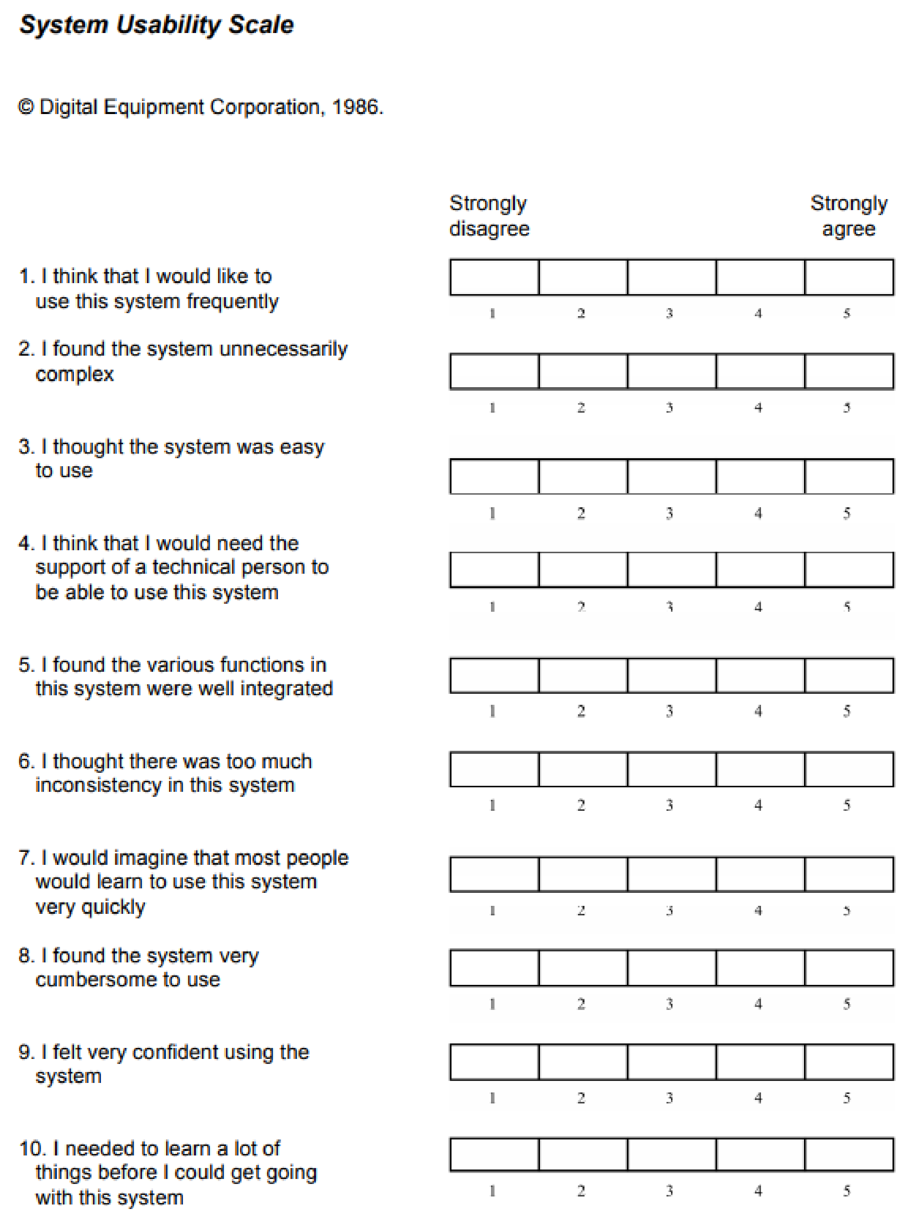
[TechnoBuffalo LLC, 2015] TechnoBuffalo LLC (2015). `https://www.technobuffalo.com/wp-content/uploads/2015/11/Xbox-One-Dashboard-1-1280x720.jpg`. Retrieved August, 2017.

[Thorsen, 2010] Thorsen, T. (2010). Wii sales near 71 million, DS almost 129 million. *Gamespot, May*, 6.

[Weichert et al., 2013] Weichert, F., Bachmann, D., Rudak, B., and Fisseler, D. (2013). Analysis of the Accuracy and Robustness of the Leap Motion Controller. *Sensors*, 13(5):6380–6393. Retrieved August, 2017.

Appendix A

System Usability Scale Questionnaire



System Usability Scale

© Digital Equipment Corporation, 1986.

	Strongly disagree				Strongly agree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5