A Framework for Multi-Device Application Development based on Kinect

Master Thesis

Elena Teunissen
<telena@student.ethz.ch>

Prof. Dr. Moira C. Norrie
Dr. Michael Nebeling, Maria Husmann

Global Information Systems Group
Institute of Information Systems
Department of Computer Science

16th August 2013
Abstract

Interactive systems set in multi-device environment continue to increase in popularity, prompting both practitioners and developers to experiment with emerging technologies. This thesis presents XDKinect — a lightweight framework that facilitates development of cross-device applications using Kinect to mediate user interactions. The main benefits of XDKinect include its simplicity, configurability and extensibility. Our framework features a time-based API to handle full-body interactions, a multi-modal API to capture speech and gesture recognition, a settings API to optimise for specific application requirements, an API to utilise proxemic awareness information, and a cross-device communication API. A user study was conducted to investigate the potential of these features in terms of effectiveness, ease of use and possible use in the future. We show several example applications of XDKinect, as well as discussing advantages and limitations of our toolkit as revealed by the user evaluations.
Acknowledgements

First of all, I would like to thank Prof. Moira Norrie and Dr. Michael Nebeling for providing an amazing opportunity be a part of a European-wide research project. It was a fascinating and challenging thesis, which left me with a happy feeling of contributing to the research community. Many thanks to my supervisors Dr. Michael Nebeling and Maria Husmann for their valuable feedback, guidance and shared excitement for this project. I would also like to thank several members of the Globis group that sacrificed an entire hour of their lives to participate in my user study only to get a tiny chocolate as a reward.

Furthermore, I would like to express my gratitude to my family and friends for their continuous support. The thesis is also dedicated to my babushka who always believed in me and dreamt of the day when I would finally get my degree. My warmest thanks go to Reto, who will probably forget to mention me in his thesis.

Finally, I would like to thank VIS for offering the wonderful coffee machine that ensured my survival at this university.
### Contents

1 Introduction ........................................... 1
   1.1 Motivation ....................................... 1
   1.2 Contributions .................................... 2
   1.3 Structure ........................................ 2

2 Related Work ........................................ 5
   2.1 Proxemic and Ambient Interaction .............. 5
   2.2 Multi-Device User Interfaces .................. 6
   2.3 Kinect in Browser Applications ............... 7

3 XDKinect ............................................. 9
   3.1 Design Process .................................. 9
   3.2 Concepts ........................................ 10
      3.2.1 Time-Based API ............................ 10
      3.2.2 Multi-Modal API ............................ 12
      3.2.3 Proxemic API ............................... 12
      3.2.4 Settings API ............................... 13
      3.2.5 Inter-Client Communication API .......... 16
   3.3 Implementation .................................. 17
      3.3.1 Architecture ............................... 17
      3.3.2 XDKinect Protocol ......................... 18

4 XDKinect Applications ............................... 21
   4.1 ScrapBook ........................................ 21
   4.2 Fotobook ......................................... 22

5 Evaluation ........................................... 25
   5.1 Method ........................................... 25
   5.2 User Study Application ......................... 26
   5.3 Results .......................................... 27

6 Conclusion ........................................... 31
   6.1 Discussion ....................................... 31
   6.2 Limitations and Future Work ................... 33

Appendices ............................................. 35
   A User Study Documents ............................. 37
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1 Background Questionnaire</td>
<td>37</td>
</tr>
<tr>
<td>A.2 Task Setting</td>
<td>39</td>
</tr>
<tr>
<td>A.3 Cheat Sheet</td>
<td>41</td>
</tr>
<tr>
<td>A.4 Post-Study Questionnaire</td>
<td>44</td>
</tr>
<tr>
<td>A.5 Coding Templates</td>
<td>47</td>
</tr>
<tr>
<td>A.5.1 Ambient</td>
<td>47</td>
</tr>
<tr>
<td>A.5.2 Main</td>
<td>49</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Motivation

The development and design of multi-device interactive systems play an increasingly major role in the research community nowadays. Innovative methods to further extend the spatial relationship between users and devices arise constantly, and so existing technologies evolve to adapt to the users’ needs. As we experience proliferation of highly interactive systems, the classic approach of resorting to keyboard and mouse as the only input feed from the environment may become superfluous. As such, tablets and smartphones are being controlled by touch gestures and some speech input. Desktop computers as well are becoming equipped with touch-based displays, and many operating systems (e.g. Windows) also integrate speech recognition techniques for interaction.

Microsoft’s Kinect for Windows\(^1\) allows for experimenting with new forms of interactions based on the spatial relationships between users and devices. Its features include support for 3D gestural skeleton tracking and speech input among others, adding a rich form of interactivity with the application. Kinect games for Xbox are already an integral part of today’s households. As motion control technology and algorithms grow more advanced and accurate, the number of people experimenting with full-body interactions increases.

To address emergence of these novel Natural User Interfaces, there is a rising trend in ubiquitous computing research to explore novel interaction practices between people and set of devices, and of devices to devices. A key requirement to embody assessment of these spatial relationships is flexibility to a variety of different settings. However, despite the increasing accessibility of hardware, most of today’s development environments are not adapted for building this kind of interactive applications. The programmer has to take care of many low-level details, which hinders their progress and makes development a tiresome and

\(^1\)www.kinectforwindows.org
cumbersome task. The fundamental research question is therefore how to design a flexible, lightweight toolkit for both researchers and practitioners that enables developing applications for a wide span of scenarios.

This thesis presents XDKinect, a toolkit for interfaces that can scale across devices using Kinect to mediate user interactions. Its skeleton, depth, and speech input channels are valuable for augmenting target applications. A further benefit is Kinect’s relatively cheap price and popularity, thus opening our research for the “general public” beyond research institutions. We focus in particular on web applications due to their richness and diversity. Our goal is to enable multi-device application developers to leverage their existing knowledge in web technologies. Additionally, we want to provide a lightweight toolkit for easy and rapid development of Kinect-based applications that can be executed in modern web browsers.

1.2 Contributions

The main contribution of this thesis is a lightweight framework to facilitate building multi-device interfaces that the users can interact with using Kinect. The toolkit features a time-based API to query and constrain Kinect streaming data, a multimodal API to configure skeletal tracking, gesture and speech recognition, as well as the inter-client communication API for cross-device applications. Furthermore, the core logic of XDKinect applications can be flexibly moved between the client and server as both components are extensible. This is particularly useful for advanced gesture recognition algorithms requiring complex machine learning techniques that would be difficult to implement on the client-side, such as the Kinect Interactions features.

The developers can customise interaction parameters to adjust to application specific requirements in order to achieve a better performance. For instance, XDKinect allows for distinguishing between different Kinect tracking modes, selecting only specific skeleton joints for tracking and setting the grammar for speech recognition.

Moreover, XDKinect features a simple version of proxemic awareness. Hall’s proxemics theory [3] discusses inter-personal distances, position, orientation and movement. XDKinect can discern between near and far mode depending on how close/far the user is situated to the Kinect sensor. This thesis also presents several XDKinect applications as proof-of-concepts, one of them illustrating a simple version of proxemic interaction [1]. The system can take cues from the user’s position and adapt the displayed content to fit either into personal or ambient interaction modality. Additionally, collaborative interaction is feasible, as XDKinect can handle input from many users. It detects the number of users currently communicating with the system, and reacts if users leave or join the application.

1.3 Structure

The remainder of this thesis is organised as follows. In the next chapter, we present the state of the art of current research in developing for multi-device environments, experiments with proxemic interactions, and web-based applications with Kinect. Chapter 3 introduces key fea-
tutes of XDKinect, discusses its architecture and provides fine points about implementation. Chapter 4 describes two sample applications developed with XDKinect. Since XDKinect’s target audience are developers, we assessed the frameworks’s usefulness and effectiveness with respect to building applications by recruiting a number of programmers. This user study is explained in detail and evaluated in Chapter 5. Finally, we compare XDKinect against other solutions and discuss limitations as well as future work.
1.3. STRUCTURE
XDKinect aims at facilitating development of highly interactive systems set in a multi-device environment. To our knowledge, there is still little research focusing at achieving our set of goals with Kinect and web applications. However, the idea of rich, highly interactive federations spanning many users and devices is not new. Therefore, many concepts and ideas from researching ecologies of devices and people as well as touch-based multi-device interactive systems can be applied to our work. In particular, the field of proxemic interaction has been explored quite extensively, resulting in many fascinating interaction technologies and applications thereof. Numerous forays have been undertaken to explore touch-based toolkits for browser interaction as well. We sampled a few papers about research in ubiquitous computing and Human Computer Interaction, which we discuss below in more detail.

2.1 Proxemic and Ambient Interaction

As stated in 1, the research in proxemic interaction investigates spatial relationship between users and objects taking cues from distance, orientation, movement, and identity [1]. The GroupTogether system [7] augments proxemic principles with theories of F-formation and micro-mobility. The former investigates physical arrangements of a small group of people engaged in a focused conversation, while the latter researches the impact of re-orienting and re-positioning physical devices on information sharing techniques. Combining these two aspects, the researchers designed new techniques of cross-device interaction with emphasis on fluid and smooth communication. For instance, tilting a device by a small angle triggered a semi-transparent representation of the selected item to appear on all devices within proximity. The recipient could either keep a copy of shared content or let it disappear once the sender would abort the sharing process.

Extending their work, [1] studied interaction behaviour of one person or multiple persons in relation to one or multiple devices in their immediate environment. They extended Hall’s proxemic theory by adding notions of fine grained sensing of nearby people and
devices, mediating between implicit and explicit interaction, and by distinguishing between four discrete proxemic zones. This logic was incorporated in a lightweight toolkit, which they illustrated on a scenario of a proxemic-aware media player. [6] describes a toolkit that gathered data from various tracking devices and released it through an event-driven object-oriented API. Relationships between entities can be observed and closely investigated using the Visual Monitoring Tool.

Similar ideas for proxemic interaction are also applied in [15], conceptualising design principles for developing sharable, interactive public and private ambient displays. These principles include visualising the data in a calm manner, meaning and functionality should be displayed naturally, short-duration fluid interaction, shared use etc.

## 2.2 Multi-Device User Interfaces

Our framework found some inspiration in jQMultiTouch [9]. Nebeling et al. contributed a multi-device web toolkit for touch interfaces, which is compatible across multiple devices and platforms. One of the key features of jQMultiTouch is a central touch event history, which can be queried and evaluated to enable better gesture recognition. The framework also boasts unifying browser-specific touch events as well as high flexibility by means of attachable behaviour, i.e. allowing developers to customise or even replace default behaviour.

Substance [2] is another example of a framework for interactive multi-device applications. Shared Substance is a middleware assisting flexible sharing of both digital and physical resources. One of the main concepts behind Substance and Shared Substance is loose coupling between data and functionality, which enables sharing at the programming language level and thus adds to a higher level of intuition for the developer. Design of a suitable architecture for the application is, however, left to the developer. Despite Shared Substance being targeted primarily at touch interfaces, the paper introduces fascinating ideas — especially the approach for sharing — for further research in flexible, distributed interactive applications.

A logical framework to alleviate development of multi-device interfaces was presented in [11]. Their work identifies several aspects of ongoing interaction methods with the aim of being able to better analyse and evaluate the current solutions. It provides an extensive overview about a wide range of various strategies in cross-platform design dimensions, such as UI distribution, UI granularity, UI migration, device sharing between multiple users and some other facets. Moreover, it offers a systematic classification of an assortment of design decisions and their shortcomings, and serves as a valuable starting point into the research about multi-device environments.

A recent paper [10] presents a platform to facilitate design and evaluation for cross-device applications both at the data model and user interface level. They introduce a concept of a cross-device session that links the concepts of user, device and shared information data. Developers can specify different scenarios of information sharing, where devices and sessions can be paired flexibly, allowing completely independent interactions, shared interactions or arbitrary mixtures thereof.
2.3 Kinect in Browser Applications

Kinect has been employed in a wide number of settings. For example, KinectFusion [4] enables creating a 3D reconstruction of an indoor scene. Latest researchers have also started to integrate Kinect data into web applications. For example, [8] enhanced living-room TV into a Kinect-based interactive device. They conducted an elicitation study, where appropriate gesture and speech commands for controlling a web browser on television were identified. Kinected Browser [5] is a toolkit to facilitate augmenting web pages with gesture and speech input. The system extends the familiar JavaScript DOM event model with custom skeleton-triggered events. Additionally, Kinected Browser provides access to the raw depth and colour stream data.

KinectJS\(^1\) is a powerful JavaScript-based framework for Kinect applications. KinectJS enables tracking 20 joints, recognises simple built-in gestures and gives access to Kinect’s raw RGB and depth data. Sadly, the project seems to be abandoned as it works only with old browsers. Moreover, KinectJS does not support development in cross-device environment. The open-source DepthJS\(^2\) project also fits into the category of early Kinect-steered browsers for Chrome and Safari. Their toolkit provides enhancements such as gesture-based tab switching, button pressing or scrolling. Developers can use the JavaScript API to access this functionality.

---

\(^1\)http://kinect.childnodes.com/
\(^2\)depthjs.media.mit.edu
The aim of XDKinect is to facilitate development of cross-device applications that can cater for a wide range of possible scenarios. Thus, we focus on developing a lightweight and extensible framework that encapsulates low-level details from the programming API exposed to the programmer. In this chapter, we first outline our design process by exhibiting a few drafts. Later we present a settings API to configure the application to fit the precise requirements, a time-based API for full-body interaction tracking, a multi-modal API encompassing gesture and audio recognition, a cross-device communication API, and a proxemic API to take cues from the users’ position and environment.

3.1 Design Process

The very first scenario that boosted our initial excitement to work on the project was the designers’ meeting room. In this scenario, a software could give some thought-provoking impulses and help sustain creativity. A collection of images for inspiration could be shown on a huge ambient display using “calming technologies”. Using Kinect’s microphone array, the application could “eavesdrop” on the conversation, pick a few selected terms and adapt the image slideshow to the suitable content. Other context dimensions such as the number of users, their identity, and position towards the sensor could be factored into the choice of content on the ambient display. People would browse through the images, select one, and pull it over to other devices, e.g. a mobile phone for further inspection. The designers’ meeting room was used as inspiration for other applications, which in turn drove the development of XDKinect and helped assess its features with respect to other requirements.

A very important step in creating a generic and useful framework to accommodate the needs of most versatile applications is identifying system requirements. In order to do so, we created multiple test applications to assess Kinect’s capabilities and recognise the most common features. For instance, one of the early samples was a single-device drawing application on the screen using the right hand as painting brush. This application laid
fundament of a time-based API and basic client-server communication. The server tracked a user’s skeleton, read out positions of the right hand joint and sent them to the client. On the client side, the application drew a line between the scaled position of the current coordinates and the previous ones. We experimented with a few formats of JSON messages with coordinates before finding the optimal one. All interaction parameters — tracked joints, Kinect tracking mode, a single user — were still hard-coded on the server.

Another example, the ScrapBook, was intended as an homage at the meeting room scenario described above. The application enabled browsing through a gallery of images, selecting one and pulling it over to another display for a closer look. The main display - storage for the “best” pictures — could host four images and responded to voice commands. This more advanced application demonstrated the facility to exchange information between devices as well as speech recognition. The ScrapBook sported an increased flexibility, as clients could already subscribe for specific joints, determine Kinect skeleton frame rate and set Kinect tracking mode. In order to improve ease of use and the quality of code, we added custom events. Prior to that, all client-server interaction occurred through the standard JavaScript web sockets API.

Inspirations from [1] drove us to add a basic proxemic API. For example, if an application included an ambient display, it could react upon changes in distance and adapt the shown content. Later we added notifications when users leave or join the application.

In March 2013, Microsoft released a new version of Kinect SDK\(^1\), which was after we started working on the XDKinect project. The most exciting novelty are Kinect Interactions\(^2\), a set of features that incorporates gesture-based tracking of user’s primary interactive hand. We extended the server-side module of XDKinect to include these features with only little effort.

### 3.2 Concepts

XDKinect follows a classic client-server architecture. The client-side module has an event-driven design. Whenever the library receives new content from the server, it fires an appropriate event. In the following, we discuss the APIs and present the corresponding events.

#### 3.2.1 Time-Based API

The Kinect device utilises the infrared camera to detect humans. It is able to recognise up to 6 people, and track two users in detail. Currently, Kinect tracks 20 joints in 3D. The \( x \) coordinate is the horizontal distance between the joint and the sensor, \( y \) corresponds to the vertical one, and \( z \) is the distance accordingly. Distances are measured in metres, and the coordinate system origin is located at the camera position. Due to some noise and measurement errors, skeletal information can appear unstable. This can result in jitters and spikes when, for example, an application would drive a cursor icon on the screen using the position of a hand joint.

\(^1\)http://www.microsoft.com/en-us/kinectforwindows/develop/new.aspx
In order to avoid jitter, XDKinect provides access to the smoothed coordinates of all joints, taking into account some resulting latency. The origin of the coordinate system is per default the Kinect position, which can be re-configured to the centre of user’s body to accommodate application’s needs. Inspired by jQMultiTouch [9], XDKinect maintains internally a history of joints’ coordinates for each tracked user. Possible applications involve custom gesture recognition, cursor tracking or various statistics to deduct users’ behaviour. Every incoming Kinect skeleton data is pushed into the buffer, and the oldest entries are deleted. The default storage duration amounts to 4.5 seconds, but it can be configured as well. Programmers are able to query it easily, as the history is implemented as a simple JavaScript array. A sample coordinates history object is shown in Listing 3.1.

```
[{skeletonId : 148, timestamp: 1374225394869, HandRight: {x : 0, y: 0.1, z: 0.2}},
 {skeletonId : 148, timestamp: 1374225395678, HandRight: {x : 0.3, y:-0.15, z: 0.25}}]
```

Listing 3.1: A sample history object. Only right hand is tracked.

XDKinect provides an event to capture skeleton coordinates from the server. The event is fired whenever a new Kinect skeleton frame is available. In order to receive skeleton data, developers must declare a listener function. Listing 3.2 shows sample code for the event and illustrates possible usage for the history.

```
XDKinect.addListener("coords", coordsEventListener);

function coordsEventListener(coords){
    var xKinect = coords.HandRight.x;
    var yKinect = coords.HandRight.y;
}

/*/ Demonstrates sample history usage */

function analyzeHistory(){
    //returns the entire history
    var history = XDKinect.getCoords();
    var oldestTimestamp = history[0].timestamp;
    //returns all skeletons for the entire history duration
    var allSkeletons = XDKinect.getAllSkeletons();
    //returns only currently tracked skeletons
    var currentSkeletons = XDKinect.getSkeletonIds();
    //calculates inactive skeletons
    var inactiveSkeletons = $.grep(allSkeletons, function(x) {
        return $.inArray(x, currentSkeletons) < 0);
    });
}
```

Listing 3.2: coords event code sample and history usage.
3.2.2 Multi-Modal API

As mentioned above, XDKinect offers an API to access recognised gestures and speech from the server. Both interaction modalities stem from the Kinect interaction stream and Kinect audio stream respectively. XDKinect supports such gestures as grip, grip release and press detection, and can discern between the right and the left hand. Analogously to the skeletal tracking, the target application must subscribe for the desired gestures. As Kinect Skeleton stream and Kinect Interaction stream are independent of each other, gesture support and joint tracking are two completely different concepts, i.e. subscribing to track the right hand will not automatically prompt Kinect to detect any gestures performed by that hand. XDKinect’s speech recognition service employs the Microsoft.Speech platform. Similarly to the previous features, target application can specify a set of words or expressions for Kinect to recognise.

A gesture event is fired whenever a subscribed gesture has been recognised by Kinect. A listener function and a list of gestures must be provided upon initialisation. The listener function is passed the recognised gesture name, since most application probably utilise more than one gesture. The logic for the speech recognition event is similar. The listener function’s only parameter is the recognised term. Listings 3.3 and 3.4 depict the corresponding code snippets.

```javascript
XDKinect.addListener("gestures", gestureEventListener);

function gestureEventListener(gestureType){
  if(gestureType === "GripLeft"){
    console.log("Grip with the left hand");
  }
}

Listing 3.3: gesture event.
```

```javascript
XDKinect.addListener("speech", speechEventListener);

function speechEventListener(recognisedTerm){
  if(recognisedTerm === "dog"){
    console.log("You said dog");
  }
}

Listing 3.4: speech event.
```

3.2.3 Proxemic API

Another feature of XDKinect is distinguishing between near and far distances between the user and Kinect. In accord with principles of proxemic interaction, the system can take cue from a user’s position and react accordingly. For example, an ambient display could distinguish between the sleep mode when users are far away, and become suitable for interaction

---

when users are in close proximity. Application can determine the distance threshold that triggers transition from one mode into the other. Concerning the programming API, moving farther away from or moving closer to the sensor are two distinct events, each of which has to be subscribed for explicitly. Listing 3.5 shows an example code for the instance of moving farther away than the specified threshold.

```javascript
XDKinect.addListener("distanceFar", distanceFarEventListener);

function distanceFarEventListener(){
  console.log("Moved farther away");
  // switch CSS layout
}
```

Listing 3.5: distanceFar event.

A further artefact of XDKinect is support for multiple user tracking. Whenever a user leaves or joins the field of Kinect view, XDKinect notifies the application. User identification is performed on the basis of the Kinect skeleton ID. However, once a user exits and re-enters the application’s field of view, Kinect assigns the same user a different skeleton ID. Akin to the previous section, appearance of a new user or departure of an existing user triggers two different events. The Listing 3.6 shows the code for the case when a new user joins the system. Code for the opposite situation is very similar. The listener function takes a Kinect skeleton ID as parameter.

```javascript
XDKinect.addListener("userJoined", userJoinedEventListener);

function userJoinedEventListener(skeletonId){
  console.log("Welcome, user " + skeletonId);
}
```

Listing 3.6: userJoined event.

### 3.2.4 Settings API

One of XDKinect’s key features is the facility to configure the server so as to receive exclusively the required content. Prior to any user interaction, all clients must connect and notify the server about their individual configuration details, such as status, tracked joints, speech grammar, and gestures. Every JavaScript file corresponding to every client of the system must declare a JavaScript object listing any dimensions for fine-tuning. A sample configuration object is shown in Listing 3.7.
The following fields are required: the URL to create a web socket, a unique ID, and number of clients. Any other configuration parameters may be provided if necessary. Table 3.1 lists all available configuration parameters and corresponding values. These parameters are sent to the server in JSON format with the appropriate header field. Once all clients are connected and individual configuration is completed, interaction can begin.

Server configuration is only possible at the initialisation phase. The motivation behind this design decision is an attempt to avoid delays when Kinect parameters will have to be reset during runtime. Switching Kinect tracking mode results in a delay of several seconds, which is undesirable for an interactive, real-time application. Analogously, reloading grammar for speech recognition also exhibits seconds-long delay. While lags for other configuration parameters are not of a magnitude of several seconds, they can still manifest in an unresponsive application.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>Yes</td>
<td>None</td>
<td>Web Socket URL</td>
</tr>
<tr>
<td>id</td>
<td>Yes</td>
<td>Empty String</td>
<td>A unique ID.</td>
</tr>
<tr>
<td>numClients</td>
<td>Yes</td>
<td>1</td>
<td>The currently supported gestures are grip, grip release and press gesture by the user’s either right or left hand.</td>
</tr>
<tr>
<td>status</td>
<td>No</td>
<td>passive</td>
<td>An active client receives joint coordinates from the server; a passive client does not.</td>
</tr>
<tr>
<td>joints</td>
<td>No</td>
<td>[]</td>
<td>A set of user joints to track.</td>
</tr>
<tr>
<td>gestures</td>
<td>No</td>
<td>[]</td>
<td>Grip, GripRelease and Press gestures.</td>
</tr>
<tr>
<td>speech</td>
<td>No</td>
<td>[]</td>
<td>A speech grammar with words or expressions for recognition.</td>
</tr>
<tr>
<td>historyStorage</td>
<td>No</td>
<td>4.5</td>
<td>Skeleton coordinates storage duration in seconds.</td>
</tr>
<tr>
<td>numSkeletons</td>
<td>No</td>
<td>1</td>
<td>The maximum number of users to interact with the system.</td>
</tr>
<tr>
<td>frequency</td>
<td>No</td>
<td>10</td>
<td>Fixes time span in milliseconds between two skeleton frames sent by the server.</td>
</tr>
<tr>
<td>trackingMode</td>
<td>No</td>
<td>default</td>
<td>Default, Seated or Near Kinect tracking mode.</td>
</tr>
<tr>
<td>distanceThreshold</td>
<td>No</td>
<td>0</td>
<td>Proxemic distance from the sensor from which the system can take cues.</td>
</tr>
<tr>
<td>isRelativeToBody</td>
<td>No</td>
<td>False</td>
<td>Indicates whether the origin of Kinect skeleton coordinates is at the camera position or at the centre of user’s body.</td>
</tr>
</tbody>
</table>

Table 3.1: Overview of configuration parameters.
The primary aim of the settings API is to render the construction of cross-device applications as convenient as possible. Furthermore, many parameters serve the goal of increasing the system’s overall performance. For example, developers may specify the maximal number of users to interact with the system. Should there be more people in vicinity than the provided parameter, their data will not be transmitted. New Kinect skeleton frames are generated with the rate of 30 frame per second — the actual rate may vary according to the CPU load, leading to the possibility of the client collapsing from a huge amount of information. In order to decrease delays on the client side, not every skeleton frame is sent.

3.2.5 Inter-Client Communication API

Last but not least, XDKinect supports cross-device communication. Information can be exchanged between any registered clients of the system. For example, referring to the scenario from the section 3.1, when a designer selects an image from the ambient display to pull it to their mobile phone for closer inspection, XDKinect would send a message from the ambient display to the mobile phone with the image data.

Upon registering with the server, each client is automatically assigned a unique random ID that is used by the server for sending all the information. The ID provided during the initialisation phase is merely for an application developer’s convenience, since the actual ID remains transparent from the programming API. After initialisation has been completed, every active client receives both the internal and the developer-provided IDs for future inter-client communication. XDKinect is scalable to support a large number of clients. All clients are completely encapsulated from each other. Interactions performed with one client have no impact on others, unless that client explicitly wishes to pass information along. The only method of sharing a state is achieved per messaging.

Another important principle is the notion of active and passive clients. Passive clients are idle whereas active clients receive Kinect data. For example, while the ambient display is alert and awaits interaction, a secondary display could be in the sleeping mode since it needs shared content to work with. A passive client is automatically “awakened” upon a message receipt. On the other hand, an active client can be explicitly rendered passive by the application. Passive clients contribute to boost in performance, decreasing the overhead and cutting information flow.

In order to be able to accept messages from other clients, a listener function for this type of event is required. The message itself can be either a simple string, like URL of a source image selected at the ambient display, or an array of strings. Listing 3.8 demonstrates the case where the obtained message is an array.

\[5^*\text{Note that the underlying Kinect hardware can only track 2 users in detail and recognise max. 6 users.}\]
3.3 Implementation

3.3.1 Architecture

XDKinect is a traditional 2-tier client-server architecture illustrated in Figure 3.1. XDKinect is implemented in C# on the server side and in JavaScript/JQuery on the client side. The server-side module is necessary, since JavaScript cannot access the Kinect hardware.

Listing 3.8: message event for cross-device communication
The main purpose of the server application is to receive and transfer Kinect data to each client as specified by that client’s individual configuration. Moreover, the server mediates the inter-client communication. Communication between the server and client modules occurs via the web socket protocol\(^6\). Web Sockets offer full duplex bi-directional communication with much less overhead than the traditional HTTP-based methods. Thus, new data from Kinect is pushed from the server whenever it is ready. Typically, new skeleton frame arrives with rate of 30 frames per second, albeit this frequency depends on CPU load. XDKinect relies on Fleck\(^7\), an open-source web socket server in C# to transmit data to the subscribed clients.

The client-side module parses and dispatches messages from the server and exposes the data to the developer through the programming API.

### 3.3.2 XDKinect Protocol

All messages, be they notifications from the server or inter-client communication, are exchanged via JSON. Message dispatching and parsing on the client side is done on a specific header field, which designates what kind of information is being passed. XDKinect server logic can be separated into four main components: handling of the skeleton stream, handling of the audio stream, handling of the interaction stream and handling of the inter-client communication. Each of these modules is designated a particular JSON message format. Listings 3.9, 3.10 and 3.11, portray a few sample JSONs.

```json
{
  "content": "message",
  "data": ["www.images.ch/image1.jpg", "animal"]
}
```

Listing 3.9: message JSON from server to client.

```json
{
  "content": "speech",
  "data": "zebra"
}
```

Listing 3.10: speech JSON with recognition result.

```json
{
  "content": "userJoined",
  "skeletonId": 234
}
```

Listing 3.11: userJoined JSON.

Kinect skeleton stream is the primary source of information, as it offers not only joint coordinates. Proxemic API modalities — distance metrics and multiple users tracking — are deduced from it as well. Essentially, the server maintains a list of tracked skeletons for each client and checks it for new or departed users by comparing the Kinect skeleton

---

\(^6\)http://www.websocket.org/

\(^7\)https://github.com/statianzo/Fleck
IDs each time a new frame arrives. If there are any changes, the corresponding client is notified. Analogously, any client subscribed for distance change notification receives a message when a user’s z skeleton coordinate is smaller or larger than their specified threshold.

Kinect interaction stream and audio stream watch for gestures and speech accordingly. Whenever a gesture or speech is recognised, the Kinect SDK fires an event. The server compares the recognition results with the criteria set by any of the clients. Should a match occur, the corresponding client is notified. Inter-client communication is implemented using the same event listening principle. This time, the server listens for message from clients. Once such an event is fired, the server parses the message and forwards it to the target client.

In spite of XDKinect not providing an explicit server-side extension mechanisms, the framework can still be extended with little programming effort. After the core logic for a new feature is implemented, a new message type has to be defined on the server side and taught how to be parsed on the client side.
3.3. IMPLEMENTATION
4

XDKinect Applications

In this chapter, we present two sample applications based on XDKinect. The first one, the ScrapBook mentioned in Section 3.1, was produced from an early draft of XDKinect. The idea of ScrapBook is to buffer items selected from a large array of images for specific purposes, e.g. furniture design. The second one, the Fotobook, is more advanced, though the use case is very similar to the ScrapBook. We describe the ideas and demonstrate the corresponding XDKinect features in the following sections.

4.1 ScrapBook

We illustrate ScrapBook by walking through a sample scenario. A web designer, Leia, searches for a collection of pictures for a customer’s website. She stands in front of a large display and browses through the image gallery by swiping left or right to go to the next or previous item (cf Figure 4.1). One of the images catches her attention. Leia extends her arms and, in one swishing motion, pulls the image to be copied to the scrapbook display (cf Figure 4.2). Satisfied with the result, Leia continues browsing for more pictures. Another one seems to satisfy her demands, and so she drags it over as well. Yet, there is something about that image that disturbs her. Deciding to fix it, Leia makes a graphical note on it using her right hand as a brush (Figure 4.3). For some time, Leia continues browsing through the gallery and buffering pictures on the scrapbook. Suddenly she changes her mind about the very first image and decides to remove it from the storage by saying the key words “delete one” (Figure 4.4).
The early draft of XDKinect offered only the time-based and parts of the multi-modal API. Custom gestures (swipe, pulling over) were achieved by comparing user’s actions against a set of templates on the client side by accessing joints coordinates over time. The beginning of the settings API was already implemented, enabling developers to register only for selected joints or setting the time span between Kinect frames. Speech recognition was available, but not yet configurable, as all commands were hard-coded on the server side. The multi-device communication API was close to the final draft except for the absence of a convenient API wrapper for developers.

4.2 Fotobook

Fotobook is based on the last draft of XDKinect and exploits all available APIs. As before, we illustrate its usage by means of a sample scenario. A professional photographer, Luke, received a customer order to design a web photo album. Because the customers are very nit-picky, Luke worked well into the wee hours of morning and forgot to shut down the system out of sheer exhaustion. When he enters his study the next morning, fresh-rested and with a cup of steaming coffee, Luke immediately notices that his wall-sized display that shows all the pictures has switched into ambient mode. The display exhibits a slideshow of all pictures, and no interaction is possible (Figure 4.5).
Once he steps closer, the display switches into an interactive mode, detecting his presence. Luke browses through the images by using his right hand as a cursor (Figure 4.6). Stopping upon an image that looks promising, Luke performs a press gesture, and that image joins the collection on the other display. The base content is on the page, so now Luke decides to add some style. He marks the first image by uttering the speech command “one” and rotates it to the right with an appropriate mid-air gesture. Furthermore, he adds the shadow and polaroid effects to the pictures by saying the corresponding commands. Meanwhile, lunch time approaches, and Luke’s friend Han comes over. When Han enters the Kinect view, the ambient display flickers shortly to announce presence of another human. Han likes the system and wants to add a background to the photo album page. He, too, selects an image from the interactive display to copy it to the current photo album page. The system notices that a different user than Luke performs this action, and places the image as background instead of simply adding it to the page. The completed page is shown in Figure 4.7.

Here, the time-based API is used for the rotating gesture. As in ScrapBook, the coordinates of user joints are compared against boolean conditions at different timestamps. The cursor’s position was calculated by mapping Kinect skeleton coordinates into browser screen coordinates every time a new Kinect frame arrived. The multi-modal API is reflected by speech
commands and the built-in press gesture to select an image. The proxemic API is used to switch between the sleeping and interactive modes — when Luke was not in the immediate proximity, the display cycled through the slideshow. As soon as Luke approached, the display switched into the interactive mode. Moreover, the system noticed the arrival of a new user, Han, by receiving information about another tracked skeleton associated with a new ID, and recognised when he pressed upon a picture.
5

Evaluation

5.1 Method

We conducted a user study to examine XDKinect’s effectiveness and ease of use. Since the framework is targeted at developers, we recruited 12 participants with computer science background. In the beginning, people were given a questionnaire to assess their prior knowledge, such as skills in JavaScript, web development, experience with Kinect both as user and developer, and interaction design. Ages ranged from 24 to 39 years old, with median age of 28. 11 participants were male, 1 was female. Table 5.1 shows a summary of background skills for all participants rated on a 7-point scale (1 being novice, 7 being expert).

<table>
<thead>
<tr>
<th></th>
<th>Median</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>28</td>
<td>29</td>
</tr>
<tr>
<td>Web Development</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Interaction Design</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>JavaScript/JQuery</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Kinect SDK</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.1: Background Skills on a 7-point scale.

The study itself comprised 5 simple tasks, which contributed to building a fully-fledged XDKinect application step-by-step. The duration of the study was planned not to extend beyond 1 hour. Each user worked alone and was free to adjourn at any moment without completing all the tasks. The procedure was executed as follows: first, the users were given a short motivation about XDKinect and were introduced to the system setup. After completing the background questionnaire, the users worked autonomously under our supervision. Finally, users filled out a post-study questionnaire, where they assessed different criteria on a Likert scale (1 being “Strongly Agree”, 5 being “Strongly Disagree”).
All participants were able to complete all tasks with almost no help from us. The minimum completion time amounted to 25 minutes, while the maximum completion time was 65 minutes. The mean time averaged to 47.5 minutes with the standard deviation of 12.5 minutes.

5.2 User Study Application

The target system for the user study was a simple cross-device application based on Fotobook (Section 4.2). It consisted of two XDKinect clients — an ambient display showing an image gallery, and another display where the selected images would be copied to (cf Figure 5.1). Users would move a cursor with their right hand and choose a picture they liked most by performing a Press or a Grip gesture. A magnified version of that image would immediately be transferred onto the main display. Participants could also test speech recognition by talking to Kinect using exclusively words from the pre-arranged grammar. The recognised test would then be shown on an HTML5 Canvas on the main display. Additionally, the ambient display would switch into the passive mode when participants were far away from it, and become interactive when people came closer. Finally, the main display showed a notification when users would either join or leave the Kinect sensor field of view.

The tasks were carefully chosen to encompass every interaction dimension XDKinect offers. For example, moving the cursor required skeletal tracking and management of the time-based API. The multi-modal API was reflected in the Press/Grip gesture for image selection as well as in audio recognition results being displayed on the canvas. The settings API was thoroughly exploited as well, since participants had to subscribe for many parameters like speech grammar, joints, gestures, and distance to the sensor. We supplied the participants with a cheat sheet describing the principles of XDKinect, configuration options, API, and all events. They had to fill the gaps in the offered skeleton code. In order to simplify the process,
we provided auxiliary functions dealing with XDKinect-unrelated functionalities like moving
the cursor. The background questionnaire can be found in Appendix A.1, the task setting is
located in Appendix A.2, the XDKinect cheat sheet and the post-study questionnaire are A.3
and A.4 accordingly. Skeleton code can be found in A.5

5.3 Results

In the post-study questionnaire participants rated ease of use and effectiveness of XDKinect
in different areas including skeletal tracking, gesture support, cross-device communication,
audio support, proxemic distance measurement, and multiple users tracking. All in all, the
received feedback was very positive. Table 5.2 shows a summary of results for all participants.

<table>
<thead>
<tr>
<th></th>
<th>Median</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ease of Use</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skeletal Tracking</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Gesture Support</td>
<td>4.5</td>
<td>5</td>
</tr>
<tr>
<td>Cross-Device Communication</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Audio Support</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Near/Far Mode</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Multiple User Tracking</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td><strong>Effectiveness</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skeletal Tracking</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Gesture Support</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Cross-Device Communication</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Audio Support</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Near/Far Mode</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Multiple User Tracking</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td><strong>Use in the Future</strong></td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 5.2: Evaluation Results for all Participants.

We also evaluated the ratings in regard to participants’ prior knowledge in various areas.
So, participants without any previous Kinect experience awarded very high marks, resulting
in both median and mode of 5 (“Strongly Agree” on a 5-point Likert scale) for all aspects.
Programmers who have at least used Kinect for a game or development were slightly less
generous. Figures 5.2 and 5.3 demonstrate the median and mode distributions of their marks.
We wondered to what extent would the users want to use XDKinect in the future considering
their previous background knowledge. However, due to different population sizes for each
group — e.g. only 4 participants crossed 3 or higher in the web development previous expe-
rience area, whereas 7 people already had skills in JavaScript/JQuery — summarising results
in a diagram is not possible. Therefore, we closely inspected only one criteria, previous Ki-
nect experience, which has groups of equal size. Median for the group with previous Kinect
experience is 4, and the group with no previous experience has median of 5, i.e. “Strongly
Agree” to use XDKinect in the future.
5.3. RESULTS

Apart from rating concepts on the Likert scale, participants were welcome to leave general feedback and suggest features for future versions of XDKinect. Whilst not everyone provided additional feedback, the received comments were very helpful for general assessment and for identifying current drawbacks. Analysis of the written feedback to both the user study and XDKinect is depicted in Table 5.3.

In summary, the overall feedback from the study was very promising. Both ease of use and effectiveness were rated highly, and 5 out of 12 participants emphasised that XDKinect was easy to use. Still, there is room for improvement. The main criticism from users was the limited gesture support for both built-in gestures and ability to define custom gestures, which is admittedly a severe shortcoming for a framework for rich, interactive systems. Currently, XDKinect only supports KinectInteraction gestures from the official Microsoft’s Kinect SDK. Gesture recognition enhancement is planned for the future work - the server-side will be extended with built-in gestures, and client-side will provide an API for user-defined gestures. Referring to Table 5.3, the other critique point was the scarce documentation (cf Appendix A.3), yet all participants were able to complete all tasks.

Finally, we analysed which current concepts and interaction dimensions had the most potential for improvement in terms of both ease of use and effectiveness. Figure 5.4 demonstrates the lowest marks for all criterias. Speech recognition and multiple users tracking received the best marks - no one awarded a lesser grade than a 4 (“Agree” on a 5-point Likert

---

**Table 5.3: Users’ comments and feedback**

<table>
<thead>
<tr>
<th>Comment</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy to use</td>
<td>5</td>
</tr>
<tr>
<td>More built-In gestures</td>
<td>3</td>
</tr>
<tr>
<td>Ability to define custom gestures</td>
<td>3</td>
</tr>
<tr>
<td>Documentation is too concise</td>
<td>2</td>
</tr>
<tr>
<td>Visual notification when Kinect detects the user</td>
<td>1</td>
</tr>
<tr>
<td>GUI on the server-side to disable certain streams entirely</td>
<td>1</td>
</tr>
<tr>
<td>Kinect for web applications is useless</td>
<td>1</td>
</tr>
</tbody>
</table>

---

**Figure 5.2: Ease of Use**

**Figure 5.3: Effectiveness**
scale). Skeletal tracking, gesture support and distance fared also fairly well with a neutral 3 being the lowest mark. Inter-client communication was the least effective but held its own in regard to ease of use. As one can conclude from the ratings depicted in Figure 5.4, the API for inter-client communication is still quite limited and could be improved in the future.

![Minimal Grades](image)

Figure 5.4: Lowest marks for each rating criteria.
5.3. RESULTS
In this thesis, we presented XDKinect, a lightweight toolkit that facilitates the development of cross-device systems supporting Kinect-based interactions. The framework is configurable and can be extended both at the client and the server side. XDKinect relies on event-driven principles, where clients subscribe for specific information from the server. We approached the task of developing XDKinect by inferring the needs of multi-device, interactive web-based systems and attempting to eliminate the major technical challenges that developers currently encounter while building such applications.

6.1 Discussion

Chapter 4 has demonstrated two example applications based on XDKinect. We are convinced that more applications will follow, since there are no limits on developers’ creativity. One of the strongest suits of XDKinect is its simplicity. As the user study has demonstrated, programmers found our API intuitive and easy to use. Since XDKinect is oriented at web environments, programmers can leverage their existing knowledge of web tools to create interactive multi-device applications with a great number of input dimensions from the environment. Despite several forays ([5], KinectJS and DepthJS open-source projects) being undertaken in the area of designing frameworks to address full body interactions, the idea of gestures as primary user input is still rarely used in practice. Existing tools served as a source of inspiration, as the number of dimensions in the spatial relationship between users and devices increases steadily. For example, [1] exploiting facets of proxemic interaction prompted us to introduce the stand-by and interactive zones for the ambient display scenario, where XDKinect takes cue from the user’s distance to Kinect. Moreover, the number of interaction modalities can be extended with only little effort due to the flexible message-exchange protocol.

We also assessed the role of XDKinect in the context of recent research. Table 6.1 presents key concepts of selected products from the selected papers. Table 6.2 lists features
and supported interaction dimensions of implementations. However, we would like to emphasise that if a feature is not supported by a toolkit, the lack thereof may not always be considered as a shortcoming. Toolkits such as Shared Substance and jQMultiTouch target cross-device applications for touch interfaces, whereas XDKinect and KinectedBrowser use Microsoft’s Kinect as interaction medium and are not designed for touch devices. It is difficult to perform a direct comparison between applications that can be constructed with these toolkits, since no framework can cater for the entire diversity of possible use scenarios.

<table>
<thead>
<tr>
<th>Feature</th>
<th>XDKinect</th>
<th>KinectedBrowser</th>
<th>jQMultiTouch</th>
<th>Shared Substance</th>
<th>GroupTogether</th>
<th>Ambient Display Framework</th>
<th>Proximity Toolkit</th>
<th>Proxemic Media Player</th>
</tr>
</thead>
<tbody>
<tr>
<td>lightweight</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>extensible</td>
<td>x</td>
<td>x</td>
<td>?</td>
<td>x</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>flexible/configurable</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>cross-device communication</td>
<td>x</td>
<td>x</td>
<td>?</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>device-device awareness</td>
<td>x</td>
<td>x</td>
<td>?</td>
<td>x</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 6.1: Main features of selected related products.

The symbol 'x' stands both for full and partial feature support for notation’s simplicity reasons, '-' denotes absence of a feature, and question mark indicates that there was not enough information about that subject. Most frameworks are lightweight, extensible, flexible and are suitable for multi-device environment. Device-to-device awareness, i.e. knowledge about position, orientation and identity of other devices in federation — is supported by toolkits aimed at principles of ambient interaction. Similarly, all listed toolkits maintain interaction from multiple users. The combination of supporting both speech and gesture input seems yet as an innovative feature, as it is incorporated only in KinectedBrowser and our toolkit. However, compared to Proximity Toolkit and GroupTogether, XDKinect offers only partial or no proxemic assistance. For example, applications based on proximity toolkits such as [1] can react on velocity and acceleration of user’s movement, leverage identity for safeguarding or media player’s personalisation, and enable four modes of proxemic awareness. Enhancing XDKinect with a richer proxemic API would allow for a number of exciting use scenarios.
### 6.2 Limitations and Future Work

While XDKinect offers a solid base for a variety of cross-device interactive applications, a number of restrictions apply. Several limitations in terms of implementation stem from underlying Kinect hardware and utilised libraries. First, gesture recognition manifests in a high rate of false positives. For example, a half-open hand is often interpreted as a grip gesture. We hope that the next release of Microsoft’s SDK will eliminate this shortcoming.

Second, XDKinect does not support a dictation mechanism for speech recognition, so the vocabulary to be recognised must be conveyed to the application in advance. As mentioned in Section 3.2.2, the multi-modal API relies on the Microsoft.Speech library. Its counterpart, the System.Speech library\(^1\) permits free text dictation. However, using System.Speech in conjunction with Kinect SDK results in error prone recognition, since only Microsoft.Speech API possesses a language pack specifically calibrated for Kinect hardware characteristics. A further restriction of XDKinect is the ability to track only 2 people in detail. One possible solution for this shortcoming would be to enhance the server-side to receive data from 2 Kinect sensors.

Other limitations address features that our basic API currently lacks. As pointed out in the user study, XDKinect offers only a restricted set of built-in gestures and no API for defining custom gestures. To alleviate this shortcoming, the server-side module could be extended to recognise a standard array of gestures, such as swipe, zoom or clap hands, and client-side component can provide an interface for user-defined gestures. Additionally, our proxemic API could be extended to accommodate modalities suggested by [1]. For instance, XDKinect could introduce several proxemic interaction zones akin to [15] as opposed to

---

to only two presently supported. User identity recognition could be augmented by using computer vision algorithms, as our existing identity mechanism is not reliable - we utilise Kinect skeleton IDs. The same user can obtain different skeleton IDs if they leave and re-enter the Kinect field of view. Kinect also provides enough data required to determine user orientation, and to supply an API. Measuring velocity and acceleration could add to some interesting future applications as well.

XDKinect employs a few design decisions that can be argued to be considered limitations. First, server settings can be configured only at the initialisation phase. The reason for that are delays that would result from changing settings at runtime. For example, Kinect tracking mode and speech grammar both take several seconds to reset, during which the application cannot take any gesture or speech input. Furthermore, use scenarios when the application would require server reset are not common. Second, clients do not share any form of session — all interactions are independent. While encapsulation leads to a simpler and cleaner API, it also restricts developers. We can imagine changing these aspects should a future, extensive evaluation classify them as being too restrictive.

In summary, XDKinect is able to support rich, interactive multi-device applications with a variety of user input dimensions. Nevertheless, there is room for improvement. Several aspects have been pointed out, but the possibilities for future work are almost endless. For instance, it would be exciting to optimise and evaluate the framework for mobile devices. An intriguing thought is combining XDKinect with other frameworks and gesture libraries; for example [12], [13], [14].

The immediate future work comprises detailed performance evaluation and elicitation of new requirements by presenting XDKinect in student- or research projects.
Bibliography


A

User Study Documents

A.1 Background Questionnaire
Background Questionnaire

Gender
☐ Female ☐ Male

Date of Birth

-----------------------------

Have you ever used Kinect before?
☐ Yes ☐ No

Please rate your skills:

<table>
<thead>
<tr>
<th>Skill</th>
<th>Novice</th>
<th>Expert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Development</td>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
</tr>
<tr>
<td>Interaction Design</td>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
</tr>
<tr>
<td>JavaScript/JQuery</td>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
</tr>
<tr>
<td>Kinect</td>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
<td>☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐</td>
</tr>
</tbody>
</table>
A.2 Task Setting
A Framework for Ambient Interaction Application Development

User Study Task Setting

System Setting

The system consists of two clients. The first client is the ambient display that hosts pictures from Flickr, which you can select and move to the main display. The main display is designed for interaction with the selected images with gestures as well as with speech commands. For this study, you are kindly asked to implement simple jQKinect applications that demonstrate some of the framework’s features.

Please solve the following tasks by completing the source code provided to you.

1. **Skeletal Tracking**
   The ambient display contains an image gallery with pictures from Flickr. Please implement a cursor following the right hand for selecting an image from the gallery.

2. **Gestures Support & Inter-Client Communication**
   When the user performs a Grip gesture with one hand as the other hand hovers above the selected image, move the selected image to the main display using jQKinect’s inter-client communication.

3. **Audio Support**
   JQKinect also supports simple speech recognition from a pre-defined vocabulary set at the initialization phase. Please select a few words of your liking (e.g. “Hello Kinect”) and write them on the HTML5 canvas element of the main display.

4. **Near/Far Mode**
   User’s distance to the Kinect device is another dimension of the application’s context awareness. In this task you should change the CSS layout of the ambient display as you move away and closer to the camera.

5. **Multiple User Tracking**
   JQKinect fires events when another user enters or leaves the Kinect view. In the final task, you are asked to print out “Bye” on the main display as a user leaves and accordingly “Welcome” as a user joins the application.
A.3 Cheat Sheet
Below is a sample config object:

```javascript
var config = {
  url: 'ws://localhost:8181/url',
  status: "active",
  id: "ambient",
  joints: ['Head', "Spine"],
  speech: ['hello', "world"],
  numSkeletons: 1,
  numClients: 2,
  frequency: 10,
  distanceThreshold: 2,
  isRelativeToBody: true
};
```

The following properties can be configured:

- **url**: URL to create a Web Socket. Required.
- **status**: An active client receives joint coordinates from the server; a passive client doesn't until it is activated. Default status is passive.
- **id**: String identifying the client that is needed for the inter-client communication. Required.
- **joints**: Set of Kinect joints user wants to track. Available joints are:
  - AnkleLeft, AnkleRight
  - ElbowLeft, ElbowRight
  - FootLeft, FootRight
  - HandLeft, HandRight
  - Head
  - HipCenter, HipLeft, HipRight
  - KneeLeft, KneeRight
  - ShoulderCenter, ShoulderLeft, ShoulderRight
  - Spine
  - WristLeft, WristRight

Spelling and capitalisation must be exactly as indicated!

- **gestures**: Gestures supported by the new Kinect Interaction SDK. Available gestures are (please mind the spelling and the capitalisation):
  - GripLeft, GripRight
  - GripReleaseLeft, GripReleaseRight
  - PressLeft, PressRight

- **historyStorage**: coords storage duration in seconds. Default is 4.
- **speech**: a set of words your application will recognise.
- **numSkeletons**: maximal number of users that will interact with the application. Default value is 1.
- **numClients**: exact number of clients your application will consist of. The default value is 2. Required.
- **frequency**: time span in milliseconds between two skeleton frames sent by the server. Default value is 10.
- **distanceThreshold**: Distance from the sensor in metres that will fire a distance event that your application may react upon. Default value is 0.
- **isRelativeToBody**: Indicates whether the origin of Kinect skeleton coordinates shall be located at the camera or at the centre of user's body (spine).

**Available events:**

- **coords**: receive coordinate message from the server for the subscribed joints. The listener function takes the latest coords as parameter.
- **speech**: speech recognized for a subscribed term. The listener function takes the recognised term as parameter.
- **message**: received a message from another display. The listener function takes message as parameter. Message can be either string or array of strings.
- **distanceNear**: user moved closer to the sensor.
- **distanceFar**: user moved farther from the sensor.
- **userGone**: user left the Kinect sensor view. The listener function takes the exited skeleton ID as parameter.
- **userJoined**: new user entered Kinect field of view. The listener function takes the new skeleton ID as parameter.
- **gestures**: Press, Grip or GripRelease gesture has been recognised. Listener function takes gesture type as parameter.
Sample coords object format:

```javascript
[{skeletonId : 148, timestamp: 1374225394869, HandRight: {x : 0, y: 0.1, z: 0.2}},
{skeletonId : 148, timestamp: 1374225395678, HandRight: {x : 0.3, y: -0.15, z: 0.25}}]
```

Example:

```javascript
jqKinect.addListener("coords", coordsEvent);
jqKinect.addListener("message", handleMessage);
jqKinect.addListener("userJoined", userJoined);

function coordsEvent(coords)
console.log(coords.HandLeft.x);
}

//here msg is an array
function handleMessage(msg){
  //outputs “image1.jpg”
  console.log(msg[0]);

  //outputs “check this out”
  console.log(msg[1]);
}

function userJoined(skeletonID){
  console.log("New User!");
}
```

Methods:

- `initialize(config)`
  Sets up the server configuration.
  **Parameters**
  - `config`: Object with server configurations

- `addListener(type, listener)`
  Adds listener function for the specified type.
  **Parameters**
  - `type`: Event type
  - `listener`: Custom function

- `getCoords()`
  Returns the entire coords history for the last 5 seconds.

- `getSkeletonIds()`
  Returns all currently tracked Kinect skeleton IDs.

- `getSkeletonCoords(skeletonId)`
  Returns all coordinates for the specified skeleton for the last 5 seconds.
  **Parameters**
  - `skeletonId`: Kinect skeleton ID

- `removeListener(type, listener)`
  Removes the specified listener function.
  **Parameters**
  - `type`: Event type
  - `listener`: Custom listener function

- `sendToAnotherClient(msg, client)`
  Sends data to another client.
  **Parameters**
  - `msg`: Single string or a JavaScript array
  - `client`: client ID as specified in the initial configuration
A.4 Post-Study Questionnaire
## Post-Study Questionnaire

1. The following JQKinect features are easy to use:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neutral</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
<th>No Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skeletal Tracking</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gesture Support</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inter-Client Communication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audio Support</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Near/Far Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple User Tracking</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. The following JQKinect features are effective for building ambient interaction applications:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Strongly Agree</th>
<th>Agree</th>
<th>Neutral</th>
<th>Disagree</th>
<th>Strongly Disagree</th>
<th>No Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skeletal Tracking</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gesture Support</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inter-Client Communication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Audio Support</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Near/Far Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple User Tracking</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. I would use JQKinect for building similar applications in the future

4. Which additional features would you recommend for future versions of JQKinect?

5. General Feedback

Thank You!
A.5 Coding Templates

A.5.1 Ambient

```html
<html>
<head>
<title>User Study - Image Gallery</title>
<link id="css-source" rel="stylesheet" type="text/css" href="../../KinectClient/css/ambient.css"/>
<script type="text/javascript" src="../../KinectClient/XDKinect.js"></script>
<script type="text/javascript" src="../../KinectClient/jquery.js"></script>
<script type="text/javascript">
    var config = {
        url: 'ws://localhost:8181/ambient',
        status: "active",
        id: "", /*TODO: Give this client a unique ID*/
        joints: [], /*TODO: Fill in the needed joints*/
        gestures: [], /*TODO: Fill in the needed gesture(s)*/
        numSkeletons: 2,
        numClients: 2,
        frequency: 20,
        distanceThreshold: 0 /*TODO: Specify a minimum distance (in meters) from the sensor for layout change*/
    };

    window.onload = setUp;

    //Subscribe for the custom events:
    $(document).ready(function () {
        //TODO: Add the listener to intercept the Kinect skeleton coordinates
        //TODO: Add the listener for the gestures
        //TODO: Add the listener as the user moves farther away from the display than the specified threshold
        //TODO: Add the listener as the user moves closer to Kinect than the specified threshold
    });

    /*
    TODO: Listener function for the coords event. Should take coords as the only parameter.
    You can use setCursor(positionLeft, positionTop) to make the cursor follow your hand movements.
    The scaleXcoordinate(kinectXCoordinate) and scaleYcoordinate(kinectYcoordinate) will return cursor left and top coordinates accordingly.
    */
```
function setUp() {
    var inc = document.getElementById('incoming');
    inc.innerHTML += "connecting to server ..<br/>";
    XDKinect.initialize(config);
    inc.innerHTML += '.. connection open<br/>';
    XDKinect.sendFlickrRequest("pictures",
        "47c69e5ca121d18460cd7ad9b5c0a10", 19);
}

function switchCSS(pathToCSSFile) {
    $('#css-source').attr("href", "../../KinectClient/" +
        pathToCSSFile);
    if (pathToCSSFile === "css/ambient_far.css"){
        doSlideShow();
    }
}

function getImageAtCursor() {
    var cursorPos = $('#cursor').position();
    var img = document.elementFromPoint(cursorPos.left - 2,
        cursorPos.top - 2);
    if (img !== null && 'src' in img) {
        return img.src;
    }
    return false;
}

function setCursor(positionLeft, positionTop) {
    $('#cursor').css({ left: positionLeft + "px", top: positionTop + "px" });
}

function scaleXCoordinate(x) {
var maxWidth = 1000;
var maxX = 0.6;
var value = ((((maxWidth) / maxX) / 2) * x) + (maxWidth / 2);

if (value > maxWidth || value < 0) return 0;
return value;

function scaleYcoordinate(y) {
    // positive values must go down, negative must go up
    y = -y;
    var maxHeight = 1000;
    var maxY = 0.56;
    var value = ((((maxHeight) / maxY) / 2) * y) + (maxHeight / 2) + 150;

    if (value > maxHeight || value < 0) {
        return 0;
    }
    return value;
}

Listing A.1: Abbreviated ambient display template code.

A.5.2 Main

<html>
<head>
    <link rel="stylesheet" type="text/css" href="../../KinectClient/css/main.css"/>
    <script type="text/javascript" src="../../KinectClient/XDKinect.js"></script>
    <script type="text/javascript" src="../../KinectClient/jquery.js"></script>
    <title>User Study - Main Display</title>
    <script type="text/javascript" src="../../KinectClient/jquery.js"></script>
</head>
<body>
<pre id="incomming"></pre>
<div id="container">
    <div id="gallery">
        <div id="cursor"></div>
        <ul id="images">
        </ul>
    </div>
    <div id="bigimagecontainer">
        <img id="bigimage" src="../../KinectClient/images/candles.jpg"/>
    </div>
</div>
</body>
</html>
```javascript
var config = {
    url: 'ws://localhost:8181/main',
    status: "active",
    id: "", //TODO: Give this client a unique ID
    speech: [], //TODO: Fill the required speech commands in here
    numSkeletons: 2,
    numClients: 2,
    frequency: 10
};

window.onload = function (){
    XDKinect.initialize(config);
    //TODO: add the listener for messages from another client
    //TODO: add the listener for the audio events
    //TODO: add the listener for the userGone/userJoined events
}

/*
TODO: Implement a listener for the client-to-client communication
Use drawImageOnCanvas(canvasId, imageSrc) to draw an image to a canvas.
You can use canvas with id "image", or make a new canvas.
*/

/*
TODO: implement a listener for the speech recognised event
Use typeTextOnCanvas(canvasId, text) to type the text onto the canvas with id "text".
*/

/*
TODO: Implement listeners for the userGone/userJoined events.
As in the previous task, use typeTextOnCanvas(canvasId, text).
*/

/ ***********************************************************/
/****************** AUXILIARY FUNCTIONS **********************/
/ ***********************************************************/

function drawImageOnCanvas(canvasId, imageSrc) {
    var canvas = document.getElementById(canvasId);
```
Listing A.2: Abbreviated main display template code.