SenScreen – A Toolkit for Supporting Sensor-enabled Multi-Display Networks

Stefan Schneegass
University of Stuttgart
Institute for Visualization and Interactive Systems
Pfaffenwaldring 5a
70569 Stuttgart, Germany
stefan.schneegass@vis.uni-stuttgart.de

Florian Alt
Media Informatics Group
University of Munich
Amalienstrasse 17
80333 Munich, Germany
florian.alt@ifi.lmu.de

Figure 1: Different sensors are used for public display application to realize input methods such as touch input or gesture input.

ABSTRACT

Over the past years, a number of sensors have emerged, that enable gesture-based interaction with public display applications, including Microsoft Kinect, Asus Xtion, and Leap Motion. In this way, interaction with displays can be made more attractive, particularly if deployed across displays hence involving many users. However, interactive applications are still scarce, which can be attributed to the fact that developers usually need to implement a low-level connection to the sensor. In this work, we tackle this issue by presenting a toolkit, called *SenScreen*, consisting of (a) easy-to-install adapters that handle the low-level connection to sensors and provides the data via (b) an API that allows developers to write their applications in JavaScript. We evaluate our approach by letting two groups of developers create an interactive game each using our toolkit. Observation, interviews, and questionnaire indicate that our toolkit simplifies the implementation of interactive applications and may, hence, serve as a first step towards a more widespread use of interactive public displays.

Keywords

Toolkits, Interactive Applications, Public Display Architecture

Categories and Subject Descriptors

H.5.m [**Information interfaces and presentation**]: Misc

PerDis'14, June 03 - 04 2014, Copenhagen, Denmark
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2952-1/14/06 ... $15.00.
http://dx.doi.org/10.1145/2611009.2611017

1. INTRODUCTION

Consumer devices, such as Microsoft Kinect[1], Asus Wavi Xtion[2], and Leap Motion[3], make it apparently easy to augment public displays with interactive capabilities, thus supporting gesture-based interaction. Prior research identified interactivity as an opportunity to increase the tangible benefit of displays for providers and users [1]. Whereas sensor data can be used to obtain information on audience behavior (e.g., how many people interacted with the display), there is also a strong benefit for the user in terms of an increased user experience and a more positive perception of such displays [17].

A major hindrance for the wide exploitation of such interactive capabilities is the fact that many providers lack the expertise to deploy interactive display applications. So far, showing static content on displays (e.g., images) is as easy as setting up a slideshow. However, the complexity is growing as interactive applications need to obtain data from sensors, analyze and process the data, and use it as an input for the application. This problem becomes even more pronounced as more sensors become commercially available.

A second challenge that arises as public displays become interconnected, ultimately forming large-scale pervasive display networks, is the need to know, which display supports which interactive capabilities. Clinch et al. argue, that in the future, public display AppStores will provide display owners with a multitude of applications to download and install on their displays, very similar to what we know today from GooglePlay or Apple's AppStore [8]. In such a scenario, applications would need to know beforehand, whether a particular display can provide the required data (e.g., a depth image, the user skeleton, or simply the number of people in the display vicinity).

[1]Kinect: `http://www.microsoft.com/en-us/kinectforwindows/`
[2]Asus Wavi Xtion:
`http://event.asus.com/wavi/product/WAVI_Xtion.aspx`
[3]Leap Motion: `https://www.leapmotion.com`

In this paper, we present SenScreen – a platform that implements a low level connection to arbitrary sensors by means of so-called adapters and streams it to a web server. Furthermore, it provides a web-based API that allows display applications to request the required data from the server. In this way, we make it possible to easily find suitable displays for an application. At the same time, we support developers of public display applications in a way such that they do not need to collect and process sensor data anymore, but can simply obtain the pre-processed data through the high-level API. To evaluate the concept we conducted a workshop. Therefore, we recruited eight web developers and asked them to implement two simple web-based games (*Whac-a-Mole* and *Pong*). Using our platform and the API, we enabled them to develop both games from scratch in less than three hours, including the implementation of the game logic and the connection to the sensors.

The contribution of this paper is twofold. First, we present the toolkit and, second, we report on the results from the workshops.

2. RELATED WORK

Our research draws from several strands of prior work, including gesture-based interaction, particularly with public displays, as well as toolkits for creating interactive UIs and collecting context data.

2.1 Gesture-based Applications

Using sensor data for gesture-based interaction has been investigated by numerous researchers. Firstly, examples include means to control TVs, for example Unicursal [4] or the MagicWand [7]. While the first one uses a touchscreen for gesture support, the latter relies on a gyroscope and accelerometer data to control IR-based devices. Secondly, in the context of mobile phones, Kratz et al. presented PalmSpace [14]. Their prototype consists of a smartphone and a depth camera that can be used to enable mid-air interaction in close proximity of a mobile phone to interact with virtual objects on the device screen. Similarly, ThumbStick is a gesture interface for the use with mobile applications [16]. It uses coloured fingernails and computer vision to determine the hand posture and hence control the virtual object. Thirdly, in the area of wearable computing, Bailly et al. presented ShoeSense, an interface that uses a Kinect attached to a user's shoe to enable gesture-based control of devices such as MP3 players [5]. WristCam is a gesture-interface aimed to recognize finger-based interaction [24]. To determine the position of the fingers, a camera is attached to the user's wrist. Fourthly, hand-gestures have been used in working environments. Wu et al. use hand-gestures to control a car robot [26]. To detect the gestures, they use accelerometer data. The Attentive Workbench is used to explore pointing gestures in a work environment [23]. All these examples show the large variety of sensors and application areas for gesture input we are going to address with our toolkit.

Today the use of sensors for interaction with public displays has become commonplace. Most popularly, the Microsoft Kinect is being used for gesture-based, playful interaction. Examples include Looking Glass, a simple gesture-based ball game used to explore the user's representation on the screen as an interactivity cue as well as the honeypot effect [18]. Strike-A-Pose is an application used to communicate how to use gestures for public displays [25]. Alt et al. used Kinect for an interactive soap bubble game to show that interaction has an effect on the users' cognition [3]. Screenfinity uses the Kinect to determine the user's position in front of the screen to visually distort the content and make it hence better readable from different positions [21]. Hardy et al. use the Kinect to investigate the effectiveness of different kinds of gestures in the context of menu selection [11]. Rofouei et al. use the Kinect and a mobile phone's accelerometer data to identify users, interacting together with an interactive surface [19]. These examples show the potential of using depth sensors for display-based interaction. However, each of the projects used their own implementation to extract and process the sensor data. We particularly aim to tackle this problem through the toolkit presented in this paper.

2.2 Toolkits for GUIs and Sensor Data

Toolkits offer means to support the application developers in various ways. Among the most popular are the Java Abstract Window Toolkit[4] or the Google Web Toolkit[5]. These toolkit, however, focus on the user interface rather than on obtaining and processing sensor data. Yet, a number of toolkits are proposed for obtaining sensor data. The Context Toolkit allows for processing a variety of sensor data and obtaining different types of context [9, 20]. The IdentityPresence toolkit allows people arriving or departing from a certain location to be determined. Holleis et al. presented the EI Toolkit which allows information from a large variety of sensors to be obtained in realtime and broadcasts them in different formats and through different communication channels [13]. Sahami et al. proposed a toolkit that allows sensor data from mobile phones to be accessed from remote phones [22]. In this way, for example, users could access the camera of the owner's phone to obtain a video stream or photo of a particular location. Nevertheless, those toolkits are mainly aimed at context acquisition but not at supporting the development of interactive applications.

Most closely related to our work are the PureWidgets [6] toolkit and the UbiDisplay [10]. The former toolkit is particularly aimed at the use for public displays, supporting both context acquisition as well as UI creation. However, it is focussing on interaction with mobile phones and has not been used for gesture-based applications that require sensor-data to be obtained, processed, and distributed in real time. The UbiDisplay toolkit partly tackles this by providing real-time sensor data. However, it is restricted to Kinect and the use on a local machine. Our work combines the advantage of both approaches, also focusing on means to abstract from the data. This is of high relevance for our work, since only few applications require the raw sensor data. Instead, it is advisable to aggregate the data to (a) decrease the amount of data, and (b) cater for the user's privacy. This first issue has been tackled by Heidemann et al. who showed how to use aggregation to decrease the amount of data [12].

In the following, we present our concept of the SenScreen toolkit with the aim to obtain information from multiple sensors, process this data, and provide it in real time to a distributed display network.

3. SUPPORTING DISPLAY NETWORKS

Research has produced a number of interactive public display applications making use of sensors, such as Kinect or Leap Motion. A common challenge for these applications is the fact that developers need to implement a low-level connection to the used sensors. At the same time, multi-display applications are scarce since the connection of multiple applications and the use of sensor data on a remote display requires significant effort.

With our work we aim to close this gap by (1) providing means for easy access to sensors via so-called adapters, (2) a server-based platform that collects, processes, and distributes sensor data, and (3) by providing a high-level API for the implementation of web-based applications using JavaScript. There are several important aspects that need to be taken into account when designing a (networked) display architecture. We address these aspects as follows.

[4]Java Abstract Window Toolkit:
`http://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html`

[5]Google Web Toolkit: `http://code.google.com/webtoolkit`

3.1 Privacy

As in almost any ubiquitous computing environment, collecting sensor information in public space creates a potential privacy threat. The use of cameras, for example, allows faces to be identified and hence users to be tracked and movement profiles to be created. In general, users do not want such sensitive information in the hand of third parties, such as advertisers or their employees. At the same time, many applications do not need this data in raw format, but an abstracted form, such as a depth image, the coordinates of the person standing in front of the display, or simply the number of persons that passed by in the last hour are sufficient. With our approach, the raw sensor data can be pre-processed and abstractions of data be made available to the requesting application. In this way, for example, applications that currently use the full resolution depth image from Kinect – like Looking Glass [18] or the SoapBubble game presented in [3] – could be modified in a way such that only the performed gestures are provided through our platform.

3.2 Unified Input Gestures

Playful interactive public display applications use a large variety of different gestures, ranging from simple hand tracking (e.g., SoapBubble [3]) to more complex gestures such as a specific user posture (e.g., Strike-A-Pose [25]). This not only puts a burden on the developer since gestures recognition needs to be implemented for each application, but also makes it difficult for the user, since there is no unified gesture set. While this is not at the focus of this research, we support the creation and use of unified gestures, which we believe in the long term to be beneficial when a user needs to understand or learn how interaction works. Note, that by requesting, for example, the skeleton data from Kinect, an application could still implement customised gestures if necessary.

3.3 Platform Independence

Interactive web applications run on a variety of systems, including but not limited to Linux desktops, Windows laptops, Mac Minis, and Raspberry Pis. To be available for as many platforms as possible we opted to implement the toolkit component responsible for obtaining the sensor information in Java. On the client side (i.e., the application using the sensor data) we opted for a JavaScript based API, to allow arbitrary web applications to access the sensor data. The information exchange is realized using web services. This ensures that adapters and clients can be created on any kind of system. Furthermore, this allows the system to be used in a local setting as well. By installing the server on a local router, for example, we can use the toolkit to create an interactive living room.

3.4 Scalability and Performance

Depending on the type of sensor, a large number of data may be created, for example, a video stream from an HD camera. Our approach to minimize the amount of data is to (a) obtain only data from the sensor a client is requesting (e.g., a video with reduced resolution and frame rate) and (b) to route and cache sensor data on a server. In this way, data can be made available to multiple clients and it can be pre-processed (e.g., downsampling, aggregation).

3.5 Ease of Use

A major hindrance for the deployment of interactive public display applications is the lack of expertise among display owners. Where static content only requires the installation of a player software, interactive applications require significant skills in programming, particularly for obtaining sensor data. With our approach, we aim to make access to sensor data as simple as installing a player (i.e., an adapter in our case). For the client we aim at providing access to the sensor data using JavaScript and, hence, allowing users with limited programming knowledge to create powerful applications. This also follows a recent trend of implementing display applications as web services to allow for an easy setup by the owners.

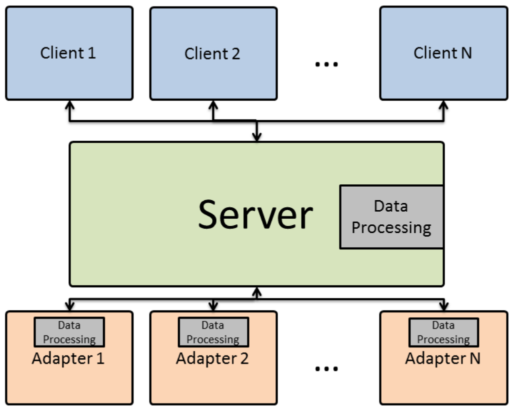

Figure 2: Overview of the architecture. The adapter and clients communicate via the server.

3.6 Payment Model

We propose a system that uses a central server being in charge of routing the whole data transferred through the network. This server is independent from the actual display stakeholders and acts as a trusted entity [2]. Since the server is in charge of controlling all data, a payment or privacy model can easily be applied. Thus, only clients that have the necessary rights can access specific sensor data. Advertisement applications, for instance, could pay for receiving anonymous profiles of the users in front of the display and adapt their content. We see particular potential in such a platform in cases where display networks employ app stores for distributing interactive applications among display owners [8].

4. THE SENSCREEN TOOLKIT

In the following section we introduce SenScreen, a sensor toolkit that allows the usage of sensors attached to nodes of display networks. The central component of the toolkit is a *server*. To this server, arbitrary sensors via so-called *adapters* can be connected. An API on the server enables *client* applications to obtain sensor data in a number of different formats. An overview of the platform is depicted in Figure 2.

4.1 Server

The main tasks of the server are, on one hand, to manage the registered *clients* as well as the *adapters* and, on the other hand, to route the sensor information from the *adapter* to the *client*. The server itself is a Java Enterprise application running on a Glassfish server. By sending a web service request, the adapter and clients can register themselves at the server. The client is then able to request a list with available adapters from the server. As soon as the client knows the available adapters, the client can request the sensor data of each adapter from the server. This is again realized through web service requests.

4.2 Adapters

The adapters are responsible for collecting data from the sensors, initially processing them, and making them available to the client via the server. They can be created in arbitrary programming languages that are needed for off-the-shelf sensors and communicate

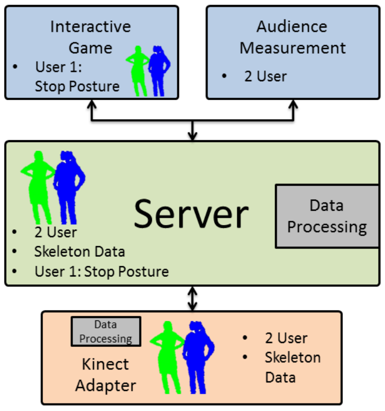

Figure 3: An example of public display applications. The Kinect Adapter records the depth image, calculates the skeleton points, and transfers the data to the server. On the server, the data is cached, gestures are detected, and both can be accessed from clients. In this example, an interactive game requests the gestures as well as the depth image and an audience measurement tool the number of users in front of the display.

via web services with the server. As one example, we implemented an adapter for the Microsoft Kinect. Data that is made available by the adapters is either raw data, such as an image or video data, or pre-processed data, such as skeleton points or recognized gestures. In addition, different properties of the data may be specified, such as the resolution of an image or the frame rate of a video.

For websites or applications to access the data, the adapter needs to register with the coordinating server and provide information on the available data and their properties. This information can then be requested via the API from the client application. As the client application requests data via the server, the latter forwards the request – including data and parameters – to the adapter who answers with the (pre-processed) data. This data is send to the server who forwards it to the requesting client application.

4.3 Client

To communicate with the server, clients can use an API. Currently, we provide a JavaScript API. Like the adapters, the client needs to register at the server and can then request information on available adapters (i.e., the available sensor data). The API provides necessary functions for registering themselves at the server as well as accessing sensor data from the *adapter*.

4.4 Data Transmission

The transmission of sensor data consists of two steps: (1) from the adapter to the server and (2) from the server to the client. All data passes through the server which routes it to the desired destination. In this way, the server can check whether the data was already requested by another client and is already available. If this is the case, data are simply duplicated on the server and forwarded to the requesting clients. As a result, adapters do not need to establish several parallel data transmissions. An example would be a display running an interactive game and an audience measurement application at the same time with both using the video stream of a camera. Each transmission is characterized by an ID. For preserving the privacy, only the server knows both IDs. We believe this to be a particular strength of the approach as in this way data can be treated in a privacy-preserving way.

5. WORKSHOP

We conducted a hands-on workshop to gain insights into how the toolkit can be used by web developers. We were particularly interested in qualitative feedback by the developers on the usability of the SenScreen toolkit. Since the main application scenarios for our toolkit are public displays, we opted to focus on the development of interactive games because we believe them to be one of the most promising applications to attract passersby. Note that our approach is of course not limited to playful applications.

5.1 Participants

In total, eight developers participated in the workshop. They were recruited through university mailing lists and posts to University bulletin boards. The workshop was divided into two sessions with four participants each. Each session took about four hours. The participants were either computer scientists or had a background in media design. From the computer scientists, all had prior experience in game development. To balance the groups, we mixed different skill sets in a way such that experienced web-developers as well as media designers where part of each group.

5.2 Procedure

After the participants arrived in the lab, we first provided them a brief tutorial to programming with JavaScript. Although most participants had prior experience with Javascript, we decided to provide a brief introduction to the most relevant aspects, so that participants could afterwards focus on the task. After the introduction, we explained them the features of the toolkit in detail. We handed out the documentation of the toolkit to assist the participants during the development task. Then, participants were presented the development task. In total, there where two tasks (i.e., two application that should be developed). In the first session, participants developed a *Whac-A-Mole* game whereas they developed a *Pong* game in the second session (cf., Figure 4). Both games should be playable using gestures obtained from Kinect data. After the participants finished developing the game, they filled in a questionnaire and we conducted semi-structured interviews.

5.3 Apparatus

For the purpose of the study we setup a server running the SenScreen toolkit and provided a laptop with an attached Kinect, which ran the KinectAdapter. For development, participants were asked to bring their own laptops and they were allowed to use their favorite IDE for the study. A local network was setup for the workshop, allowing all devices to connect. All participants were provided fast Internet access to support the design and development task.

5.4 Results

In about three hours, participants of both groups were able to solve the task and build an interactive game with gesture input using the proposed toolkit.

The *Whac-A-Mole* game was developed in an abstract manner (Figure 4 – center), with the moles being represented by circles and the hammers by squares. The moles slowly fade in. From the Kinect skeleton, the participants chose to use the player‘s hands to control the hammer. To attack a mole, players have to move their hands closer to the screen. For each mole, players receive points.

For the *Pong* game (Figure 4 – right), the skeleton point of the user's left hand is used as well. Thus, the user controls the paddle by moving the hand up and down. The bars are moved upwards when the hand coordinate is larger than the shoulder coordinate of the skeleton (i.e., players are lifting their arms above shoulder level) and downwards when the hand coordinate is smaller, respectively.

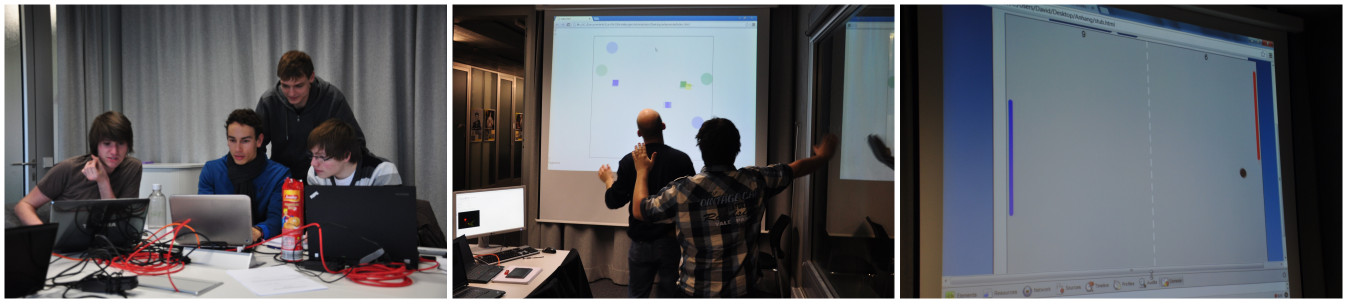
Figure 4: Participants during the workshop (left) and the resulting games: an abstract version of Whac-A-Mole (center) and Pong (right).

In the questionnaire we were interested in how difficult participants perceived the different subtasks, particularly the use of the API, developing the game logic, implementing the controls, designing the UI and an overall rating. We used a 5-Point Likert scale, ranging from 1 (very easy) to 5 (very difficult). The results show that the usage of the API is perceived as easy ($Mdn = 2$), similar to the development of the game logic ($Mdn = 2$), the user interface ($Mdn = 2$) or the mapping of the user interaction to the game logic ($Mdn = 2.5$). The results of the NasaTLX (0 = no task load, 100 = high task load) revealed a score of 31.7 for the Pong game and 45.0 for Whac-a-Mole, suggesting a low to moderate task load.

In the interviews, the participants agreed that the toolkit is easy to use. Suggestions on how to improve the toolkit included the recommendation for more detailed debug messages, means to easily visualize the skeleton data, and an opportunity to sort the skeleton data according to the position of the user in front of the screen (e.g., from left to right). Furthermore, the participants saw potential in recognizing gestures on the server and making the gesture available through the API. One participant raised privacy concerns because of sensor data (e.g., depth image of a Kinect) being transferred via a server, although the application is running locally. Regarding possible use cases, most participants stated that entertainment applications (i.e., games) in public are indeed very suitable. The use for applications deployed at home was also considered useful.

6. DISCUSSION AND CONCLUSION

In this work we presented SenScreen – a toolkit that supports developers of interactive public display applications by collecting data from arbitrary sensor sources and making them available via a high-level API. The toolkit provides so-called adapters that implement the low-level sensor connection and route the sensor data to a server that can aggregate and distribute the data. Hence, authorized clients can simultaneously access the sensor data. In a qualitative study with 8 participants we showed that our toolkit is easy to use. Two groups of 4 participants each were able to rapidly design and develop two gesture-controlled games based on data from the Kinect with basic knowledge in web development using JavaScript.

We believe our toolkit to serve as a good basis for the future development of interactive public display applications, particularly in a networked setting where multiple applications access the same sensor on a display or where games run across multiple displays. While we initially focussed on the usability of the toolkit, discussions with the participants allowed us to identify further aspects to investigate. We believe the processing and distribution of data to be one crucial point. Whereas the toolkit provides means to aggregate and abstract from the data, this is not communicated yet to the user. This might be useful, since users could be made aware of that their data is being treated in a privacy-preserving way. We plan to further investigate this in follow-up studies to increase the willingness of users to participate and to also understand the view of developers, display owners, and advertisers, who might be interested in more fine-grained data.

In the future, we plan to extend the number of adapters to allow for hooking up additional sensors, including but not limited to Leap Motion or the EyeX eye tracker[6]. Furthermore, we plan to build adapters that run on the user‘s devices rather than on the display. One possible area of application is equipping wearable computing devices such as smart textiles with adapters and, thus, allow users to interact with the display using their own gadgets. These sensors provide more possibilities to interact with display applications. In contrast to consumer devices, there are many electronic platforms available that can be used to create sensors for display networks [15]. Creating an adapter that allows for easy deployment of self-build sensors may increase the possible interaction techniques. In a further evaluation, we plan to deploy the toolkit for our University testbed and use it as a basis for courses in ubiquitous computing.

7. ACKNOWLEDGEMENTS

We thank Thomas Kubitza for his valuable input on the framework architecture and Markus Fischer for his help with the implementation of the toolkit. The research leading to these results has received funding from the European Union Seventh Framework Programme ([FP7/2007-2013]) under grant agreement no. 600851.

8. REFERENCES

[1] Alt, F., Müller, J., and Schmidt, A. Advertising on Public Display Networks. *IEEE Computer 45*, 5 (may 2012), 50–56.

[2] Alt, F., and Schneegass, S. A conceptual architecture for pervasive advertising in public display networks. In *Proceedings of the 3rd Workshop on Infrastructure and Design Challenges of Coupled Display Visual Interfaces (Capri, Italy), PPD*, vol. 12 (2012).

[3] Alt, F., Schneegass, S., Girgis, M., and Schmidt, A. Cognitive effects of interactive public display applications. In *Proceedings of the 2nd ACM International Symposium on Pervasive Displays*, ACM (2013), 13–18.

[4] Aoki, R., Ihara, M., Maeda, A., Kobayashi, M., and Kagami, S. Unicursal gesture interface for tv remote with touch screens. In *Consumer Electronics (ICCE), 2011 IEEE International Conference on*, IEEE (2011), 99–100.

[5] Bailly, G., Müller, J., Rohs, M., Wigdor, D., and Kratz, S. Shoesense: a new perspective on gestural interaction and wearable applications. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, ACM (2012), 1239–1248.

[6]EyeX: `www.tobii.com/en/eye-experience/eyex/`

[6] Cardoso, J., and José, R. Purewidgets: a programming toolkit for interactive public display applications. In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems*, ACM (2012), 51–60.

[7] Cho, S.-J., Oh, J. K., Bang, W.-C., Chang, W., Choi, E., Jing, Y., Cho, J., and Kim, D. Y. Magic wand: a hand-drawn gesture input device in 3-d space with inertial sensors. In *Frontiers in Handwriting Recognition, 2004. IWFHR-9 2004. Ninth International Workshop on*, IEEE (2004), 106–111.

[8] Clinch, S., Davies, N., Kubitza, T., and Schmidt, A. Designing application stores for public display networks. In *Proceedings of the 2012 International Symposium on Pervasive Displays*, ACM (2012), 10.

[9] Dey, A. K. *Providing architectural support for building context-aware applications*. PhD thesis, Georgia Institute of Technology, 2000.

[10] Hardy, J., Ellis, C., Alexander, J., and Davies, N. Ubi displays: A toolkit for the rapid creation of interactive projected displays. In *The International Symposium on Pervasive Displays* (2013).

[11] Hardy, J., Rukzio, E., and Davies, N. Real world responses to interactive gesture based public displays. In *Proceedings of the 10th International Conference on Mobile and Ubiquitous Multimedia*, MUM '11, ACM (New York, NY, USA, 2011), 33–39.

[12] Heidemann, J., Silva, F., Intanagonwiwat, C., Govindan, R., Estrin, D., and Ganesan, D. Building efficient wireless sensor networks with low-level naming. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, ACM (New York, NY, USA, 2001), 146–159.

[13] Holleis, P., and Schmidt, A. Makeit: Integrate user interaction times in the design process of mobile applications. In *Pervasive Computing*. Springer, 2008, 56–74.

[14] Kratz, S., Rohs, M., Guse, D., Müller, J., Bailly, G., and Nischt, M. Palmspace: Continuous around-device gestures vs. multitouch for 3d rotation tasks on mobile devices. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, ACM (2012), 181–188.

[15] Kubitza, T., Schneegass, S., Weichel, C., Pohl, N., Dingler, T., and Schmidt, A. Ingredients for a new wave of ubicomp products. *IEEE Pervasive Computing 12*, 3 (2013), 5–8.

[16] Man, W. T., Qiu, S. M., and Hong, W. K. Thumbstick: a novel virtual hand gesture interface. In *Robot and Human Interactive Communication, 2005. ROMAN 2005. IEEE International Workshop on*, IEEE (2005), 300–305.

[17] Müller, J., Alt, F., and Michelis, D. Introduction to Pervasive Advertising. In *Pervasive Advertising*, J. Müller, F. Alt, and D. Michelis, Eds., Springer Limited London (2011).

[18] Müller, J., Walter, R., Bailly, G., Nischt, M., and Alt, F. Looking glass: a field study on noticing interactivity of a shop window. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2012), 297–306.

[19] Rofouei, M., Wilson, A., Brush, A., and Tansley, S. Your phone or mine?: fusing body, touch and device sensing for multi-user device-display interaction. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, ACM (2012), 1915–1918.

[20] Salber, D., Dey, A. K., and Abowd, G. D. The context toolkit: Aiding the development of context-enabled applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '99, ACM (New York, NY, USA, 1999), 434–441.

[21] Schmidt, C., Müller, J., and Bailly, G. Screenfinity: extending the perception area of content on very large public displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2013), 1719–1728.

[22] Shirazi, A. S., Winkler, C., and Schmidt, A. Sense-sation: An extensible platform for integration of phones into the web. In *Internet of Things (IOT), 2010*, IEEE (2010), 1–8.

[23] Sugi, M., Nikaido, M., Tamura, Y., Ota, J., Arai, T., Kotani, K., Takamasu, K., Shin, S., Suzuki, H., and Sato, Y. Motion control of self-moving trays for human supporting production cell ?attentive workbench? In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, IEEE (2005), 4080–4085.

[24] Vardy, A., Robinson, J., and Cheng, L.-T. The wristcam as input device. In *Wearable Computers, 1999. Digest of Papers. The Third International Symposium on*, IEEE (1999), 199–202.

[25] Walter, R., Bailly, G., and Müller, J. Strikeapose: Revealing mid-air gestures on public displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, ACM (New York, NY, USA, 2013), 841–850.

[26] Wu, X.-H., Su, M.-C., and Wang, P.-C. A hand-gesture-based control interface for a car-robot. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, IEEE (2010), 4644–4648.