ulm university universität ulm

Ulm University | 89069 Ulm | Germany

Faculty of
Engineering and
Computer Science
Institute of Media Informatics

Motorized Tablet Holder

autonomous and self-actuated holder for tablets

Fabian Schwab (fabian.schwab@uni-ulm.de)

2014

Contents

1 ABSTRACT

In this project a motorized tabled holder was realized in hard and software to explore volumetric data. In addition, it is useful for the research of new interaction techniques. This short documentation gives an overview of the construction of the holder module and the implementation of the framework. The introduction chapter explains the basic idea why this project was realized. Also the goals of this work are explained. The subsequent chapter shows the related work which have some commonalities with this work. In the concept chapter different types of movements and the logic, how to align the tablet in space, are described. Also conceptual interaction techniques are presented in this chapter. In the next chapters the construction of the hardware is shown and described in detail and the software implementation will be explained. A sample application helps to understand the single parts of the framework and how they work together. Finally there are some limitations which are explained in the last chapter.

2 INTRODUCTION

2.1 Problem Definition & Motivation

Volumetric data is difficult to represent on a regular, two-dimensional display and thereby its difficult for humans to recognize these three dimensional structures on a regular display. To get the best impression of a volumetric data it would be useful to open up the third dimension. With the possibility to interact physically with a system which interacts with virtual structures, volumetric data can be explored more intuitive and leads to a better perception. For example a user can change his point of view on a virtual object by walking around it not by turning it on the display. So this gives the user a better and more realistic representation of the objects he works with.
There are still problems which need to be addressed. How to open up the third dimension to explore volumetric data as described above. From real holograms, as we know from the science fiction, we are still far away. Technology should be used which is already available and not in experimental stat. The most natural would be another display that hovers over an underlying one. This is not yet possible or it requires a considerable effort. Among other things, this work implements a usable hardware framework to simulate a floating display, on which new interaction techniques can be explored. Therefore, this work serves to technological aids. The main objective of the project was to develop a prototype and a framework on which future projects can be build easily.

2.2 Project Goal

On a previous project a prototype was built which allows to mount objects and let them hover over a tabletop in three dimensions. The goal of this work is to realize a self-actuated display

module which can be mounted on the previous work. The construction which holds the tablet should move as freely as possible. It's already possible to go to any three dimensional point over the table but until now, there is no way to let the display rotate automatically. With this two systems combined, it should be possible to go to any three dimensional point in the working space over the tabletop and change the orientation from the tablet. So it is possible to reach any point of view within the working space and this should help to explore volumetric data more precise and get a better total overview. Next to the hardware prototype a software framework must be developed to allow developers create new applications very easily.

3 RELATED WORK

3.1 Hovering Objects And Displays

A similar approach is the project *ZeroN* [3]. In this paper, an object is also hovering and free movable by a trolley on a table. However, in contrast to this technology, the object may not be to heavy, and must be magnetic because, for the motion on the Z axis an electromagnet is used. Also in *The Actuated Workbench* [6] a magnetic object is moved by four small electromagnet. But the movement is limited to a two-dimensional surface. Alternative ways to let an object hover in the room are presented by Alroe et al. [2] as well as Marshall et al. [5]. In *Aerial Tunes* [2] the object is hovering by a continuous flow of air from the bottom of the room. The positioning is limited only to the Z-axis and can be controlled by the intensity of the air stream. In *Ultra-Tangibles* [5] even several objects can be moved simultaneously. Instead of an air stream, the objects are moved on a flat surface by ultrasound. Commercially used technologies are found in modern sports stadiums and at larger events. Thus, for the camera work of many sports and big events often *spidercams* are used. The camera is suspended from four cables and is driven by four motors in the space. At the BMW museum in Munich is another interesting approach to see. *Kinetic Sculpture* [1] is implemented by 714 metal spheres, which are attached to cables. Each individual sphere can be driven by a separate step motor and are aligned in height. This type of *array* objects can be represented in space and simulate movements.

3.2 Self-moving Objects On Interactive Surfaces

In contrast to the earlier work presented in the user *Tangible Views for Information Visualization* [10] the users need to move the object itself across the table. The object in this case

is a rectangular white surface which is irradiated from above by a projector. The display surface of the table can be extended in this way on the surface of the object and interactions can be executed by motion and tilt and rotation of the object. Advanced is this principle in the works *Use your head* [9] and *Spatially Aware Tangible Display Interaction in a Tabletop Environment* [8] from Spindler et al., however, additional cameras were used to detect the position of the users head. Thereby an additional spatial depth can be suggested to the user. In *Boom Chameleon* [11] the two previously introduced techniques are extended. Instead of an object, which is irradiated from a projector, a tablet is used. This is fastened to a gripping arm and can be moved by the user in all directions. The user have to move the tablet by himself but the user can move it freely in the space without holding it anymore after the user finds the right position. So the user can move in a virtual three-dimensional space.

4 CONCEPT

In this chapter the concepts regarding the positioning and orientation of the tablet, the interaction with the system and the modularity are explained.

4.1 Positioning And Orientation Of The Tablets

The degree of freedom is the number of movement possibilities with which an object can move and rotate independently in space. The tablet can move along the X-, Y-and Z-axis, rotate and tilt by means of a bracket around its own axis. So that the user interaction can be guaranteed at any point, the tablet must be able to reach any position along the X, Y an Z-axis of the underlying tabletops. Additional every orientation of the tablet must be possible, so that the display can always face towards the user. Along the Z-axis, the system must be able to accomplish a sufficiently large range of motion, thus allowing interactions with standing and sitting users. The tablet can be controlled by the system and the user himself by three different types of movement. In the following, these modes of motion are outlined in more detail.

4.1.1 Autonomous Movement

The tablet moves completely independently, without receiving an input from the user. For example the system can respond to an event and provide the user with an issue in relation to its current location or illustrate movements. This sort of movement could be found in a central control center. An employee monitors the status of a system, which is visualizes him a kind of blueprint on a tabletop. If there occurs a critical event somewhere in the system, the tablet can move independently to this point and bring up more information without losing

the overview. With the autonomous rotation, for example, the system can simulate the point of view of a camera in a room and show a live steam on the tablet.

4.1.2 Semi-autonomous Movement

The tablet moves independently to a position which had to be previously selected by the user. This can be done for example by a touch input on the tabletop or by a location selection from a predefined list.

4.1.3 Manual Movement

In the manual motion control of the tablets, the user enters the position and orientation of the tablet, the tablets frame or surface before explicitly. In this scenario the user anytime full control of the system but on the other hand, he has to control the system permanent and can not focus on other tasks. A manual movement of the system can be carried out, for example, by a virtual controller that appears on the tabletop. Also possible is a control by pre-defined gestures on the tabletop surface or by touching the frame.

4.2 Interaction

In this section, conceptual interaction techniques are presented. New in this project is the manual interaction over the frame in which the tablet is mounted. if the user touches the frame the system should react as expected by the user. For example, if the user wants to move the hovering display backwards, he would grab the frame and push it backwards. The system should recognize this and start the motors in the right direction before any damage occurs through the users force. With other directions on the X, and Y-axis the gestures should be as simple and intuitive as the one explained above. It becomes more complicated when it comes to the Y-axis and turning the tablets about its own axis. For this purpose it must be possible, that gestures can be implemented. One possible gesture, to turn the tablet around its X-axis, can be to touch the frame on two different points, like back and

front, at the same time. These gestures are not specified. It is left to the developer, how the motions are realized. For this it should be possible to access this part of the framework easily and by sending simple commands the motors should start and stop.

4.3 Modularity

During the project several modules should be built, in which related actions can be encapsulated. So, for example, communication, connecting various devices together, constitute a separate module. Also, the screen orientation and the position determination should be modular. Due to this modularity of the project, a base is given to the developer to quickly and easily create new applications. This provides the possibility to explore new approaches to interaction and usability of such systems.

5 IMPLEMENTATION

In this chapter the hardware and especially the software realization will be addressed. In the hardware part the construction of the frame of the self-aligning tablet holder is described and all technical aspects will be discussed. Figure 5.1 shows latest prototype as it was at the end of the work. The software part includes the frameworks emerged in this work and the frameworks which are already available and also used for this work. In particular some sample code will show how to use the framework. The open source project also includes some sample applications, which can be used to get familiar with the framework.

5.1 Hardware

Based on the existing prototype, a self-aligning tablet holder was developed (figure 5.1). In this section the main aspects of the new module are explained. Among these are the materials used, the sensors, already existing hardware and software platforms and the power supply to make the module self-sufficient. To build a lightweight but sturdy frame, mainly aluminum was used. The attachment to the telescopic rod of the carrier-system, *Beyond Surface* [4], is via a slot. This mechanism, shown in figure 5.2, narrows at the bottom part to prevent the module from falling on the table. To prevent the module form pushing too far to the back, a pin stops the module in the slot and finally an adjusting screw allows it to fix it in its final position. Under the fastening is a cylinder, also seen in figure 5.2, which includes a motor and a gearbox. The whole construction rotates around this vertical axis without any restrictions how often the tablet can be rotated. To be more precise the parts of the construction in figure 5.2 are inked for a better understanding. The dark green plate is fixed with the turquoise mounting point and locked with the adjusting screw. This dark green plate is rigid connected to the gearwheel below the red inked axis. So this gearwheel didn't turn at all. If the motor, which is inked in a gold color in the lower part of the housing,

Figure 5.1: Overview of the module

starts spinning, the motor and the gearwheel connected to it turns around the red inked axis. Responsible for the rotation around the horizontal axis is also a motor with a gearbox, which is mounted on the left side of the construction (figure: 5.3). There the motor moves the light green part of the frame which is responsible for holding the tablet in place. Both motors are attached to the frame so that the tablet can rotate without any restrictions to the number of rotations. To provide power supply for both motors, a battery is mounted on the frame. With its own energy supply there is no need of additional cables, that would twist, from the carrier-system. To control the motors and to provide a wireless communication between them and the mounted tablet, a *IOIO-OTG* board from the manufacturer *SparkFun* is used. The communication between the components is realized via *Bluetooth*. In order to use a bluetooth dongle with the USB Type A plug, a small modification had to been made to the original board. The soldered USB Micro-A socket was removed and replaced with a USB Type-A socket. Without any adapter cable the form-factor of the controller-box (figure 5.4) decreases. Further a small chip is installed (seen on the left in figure 5.4), which passes the energy form the battery to the motors in dependence which pins are turned on. For the exact measurement of the tablets orientation the internal sensors of the tablets were not used, because they are not accurate enough. For a more precise orientation measurement

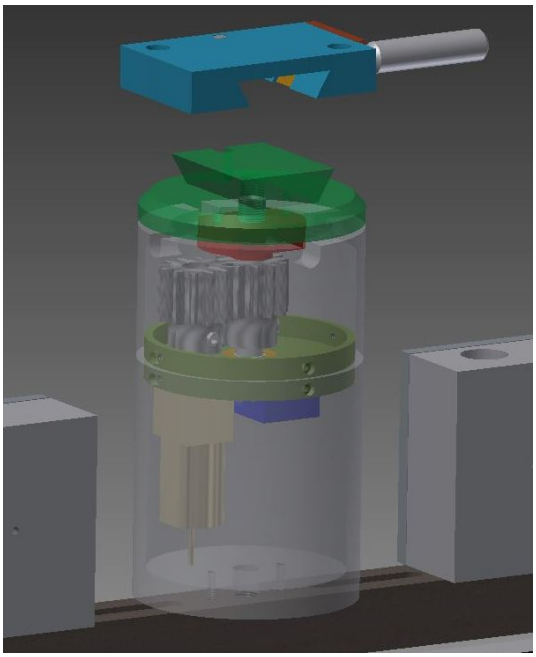

Figure 5.2: CAD view of motor, gearbox and mounting point.

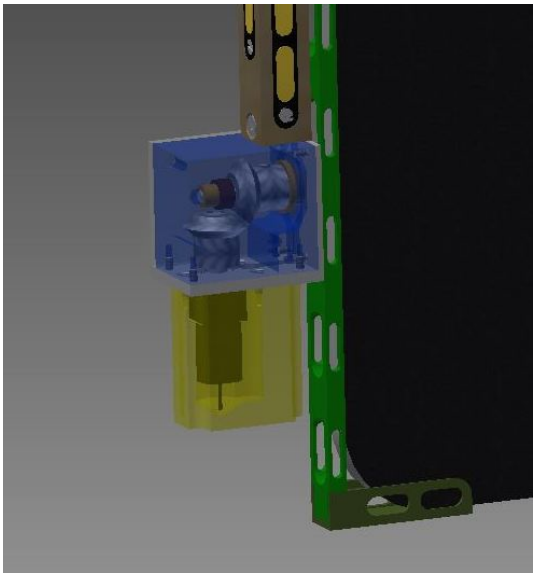

Figure 5.3: CAD view of motor, gearbox and mounting point.

a external inertial measurement unit (IMU), the *x-IMU* from *x-io Technoliogies*, is used. This device provides a variety of data but in this case only the quaternion data packets are used

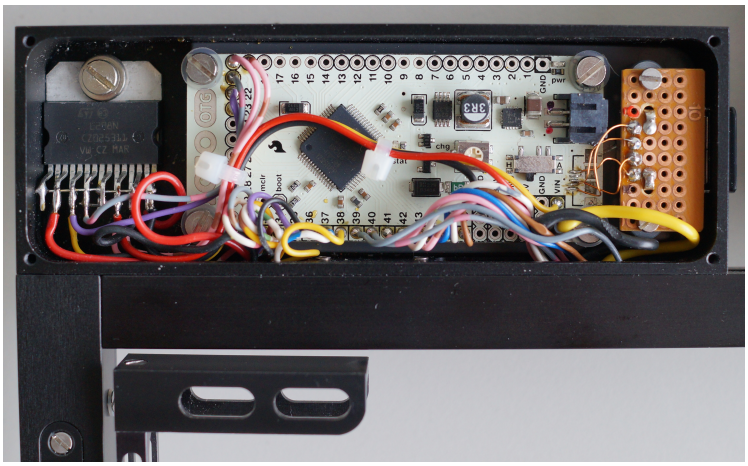

Figure 5.4: The slightly modified *IOIO-OTG* board with the additional controller chip.

and they are sampled with an update rate of 256 Hz. The tablet which is used in this project, is a *Nexus 10* from *Samsung* with the latest *Android* version. For a better interaction with the tablet and the whole system, sixteen capacitive buttons are installed around the frame. On each side of the tablet are eight of them. On the front and the back are three and on the side are two. With this symmetric construction it doesn't matter which of the sides is defined as the front. The buttons are made of brass, gilded admitted in plastic. The plastic frame isolates the buttons form one another and from the aluminum frame.

5.2 Software

In the software part of this chapter all of the frameworks, which are used or created, would be explained. Further a sample application is coded to show the most importent parts for creating a new application and to use the module easily. At first, the new created Framework which control the whole module is shown in detail and every additional frameworks used are explained too. After that all essential functions are explained and sample code will also show how to communicate between the carrier-system server and the tablet application.

5.2.1 BlueMotionLib Framework

The *BlueMotionLib* framework is divided into several packages. Among these similar tasks, such as dealing with the external IMU data, are encapsulated. In the forefront all Bluetooth-devices must be paired with the tablet over the *Android* built in Bluetooth system preference. When the *x-IMU* and the *IOIO-OTG* board are paired with the tablet, they get a name in the system preferences, which can be very helpful for finding that devices later. The central component for creating a new application with the *BlueMotionLib* is the *MotionManager* object. With the help of this object the motors of the module can be controlled and capacitive buttons can be used, but more of that later. Also all sensor data the module provides can be accessed directly. To generate the *MotionManager* object two parameters are required. Both of them are from the Java *BluetoothDevice* type and must point to a *x-IMU* and the *IOIO-OTG* board integrated in the module. If there are several paired *x-IMU* or *IOIO* boards and the developer is not familiar with Bluetooth, the right device can be found via a instance of the *DeviceManager* class. In this case only a part of the specific device name must be known. With the help of this object the developer needs to know nothing about Bluetooth and the the Bluetooth capabilities of the used device because all this is handled by the *DeviceManager*.

Devicemanager

With the *DeviceManager* it is possible to access and identify Bluetooth-devices without knowing anything about Blueooth in Java. By creating an *DeviceManager* object the first things which are checked on the device are Bluetooth availability and if the device owns a Bluetoothmodule it checked if its enabeled. If there are any problems an exception will occur with the specific text. For example an *intent* forces the user to turn on Bluetooth. The *getDeviceByName* function allows to find a paired Bluetooth-device by name and returns an *BluetoothDevice* object if found. It does not require the full name of the device to get a result. For example if there are more IMUs paired (*x-IMU 2456, x-IMU 9562*) and the first one is the target device, only *2456* as input value is sufficient to find it. So its much easier to hand all devices and create a *MotionManager* object.

Ximulib

The *ximulib* package contains all necessary classes to build up, hold and tear down a connection between the application and the passed *x-IMU* device. Every data which the application receives form the *x-IMU* is processed and prepared in this package and then passed as Java object to the next higher layer in the framework hierarchy. The original library for the communication with the *x-IMU* is written in *C++* and provided as open source project from the manufacturer on *Github* [12]. So the basics could be converted into Java but a view *C++* specific constructs were replaced with new written Java code. Further a listener interface was created for different types of payload like *command data* or *quaternion data* to provide better flow of information. When the *x-IMU* sends new data or commands via Bluetooth, is this data processed in the *XIMUDataProcessing* class an transformed in either data or command objects. To know which type of packet is received, the bytestream has to be analyzed byte by byte. Unfortunately the byte-protocol is not documented very well so it takes his time to get into it. If an observer is registered, normally the *MotionManager* object, every time the *x-IMU* gets moved new data is available for further processing.

Ioiolib

This package encapsulates all functional tasks in context with the *IOIO-OTG* board embedded in the module. For example tasks like controlling the motors to change the module orientation or any tasks with the capacitive buttons mounted on the frame. For the capacitive sensing there are sixteen buttons arranged on the frame. Every button has a unique identification number and owns two states *ENTER* and *LEAVE*. The two states are Java *ENUM*, whereby *ENTER* is triggered every time the capacitive resistance secedes and when its going back to normal *LEAVE* is triggered. By implementing the *TouchListeners* such events can be received. Next to the status information the button id is included in a notification, so that each button can be assigned individually. The button id is numbered consecutively from zero to fifteen in which zero to seven is located on the left side of the frame and eight to fifteen on the right. By a corresponding implementation gestures can be realized too. It is also possible to adjust the sensitivity of the capacitive surfaces so that an accidentally touched button can be ignored in the lower level of the processing hierarchy and no further code based solution is necessary. With the *setSensThreshold* method from the

manager object, a threshold can be set for each button individually by passing the threshold value and the id of the button. The default value (100) gets the best results in most cases. For controlling the motors, there are also different states. With the *ENUMs FORWARD, BACKWARD* and *STOP* the rotation direction of an axis are specified, where *PITCH* and *ROLL* determine the axis. Thus, the axes can be controlled independently.

5.2.2 External Libraries

Libraries that are not included in the framework but are essential for a error-free operation, are briefly explained here. For detailed information the original documentation should be used. The main libraries are the IOIO libraries. The *IOIOLibAndroid* contains the processing logic and the *IOIOLibBT* library is responsible for the communication via Bluetooth. This libraries are available as open source repositories on the manufacturer page on *Github* [7]. With the *LeviTabLib* it is possible to send and receive *JSON-Objekte* over wireless LAN. For an executive use of the self-aligning module with an *Android* device this library is not required. To read more about the scope of functions of the *LeviTabLib* the documentation from the *Beyond Surface* [4] project can be used.

5.3 Example Application

In this section the basics are explained how to create an application with the *BlueMotion* framework. The goal of this application is to use the motor controls and make use of the capacitive buttons. Further the orientation data were received by register on the frameworks listener. The first step is to create a new *Android-Project* in *Eclipse* and import the *BlueMotionLib, IOIOLibAndroid, IOIOLibBT* and *LeviTabLib* libraries. To be notified about upcoming events the *MotionListener* has to be implemented, as in listing 5.1.

Listing 5.1: *MotionListener* for orientation changes and touch buttons

```
1 public class FullscreenActivity extends Activity
2         implements MotionListener
3 {
4         ...
```

```
5           @Override
6           public void buttonStatusChanged(int id, ButtonState s) {
7                   ...
8           }
9           @Override
10          public void orientationChanged(float roll,
11          float pitch, float yaw) {
12                  ...
13          }
14  }
```

To create a *MotionManager* object a *x-IMU* and a *IOIO-OTG* board must be passed to the constructor. This step lies with the developer because it is possible that there are more that one *x-IMU* and *IOIO-OTG* boards included within one project. It's also possible to create more than one manage object to manage all the devices. The *MotionManager* object gets the nedded devices as Java *BluetoothDevices* objects. If the developer do not know how to handle Java Bluetooth, it is much easier to use the *DeviceManager* to get an object of the needed type and the right device. To search for a device the name, which the system provides after pairing, can be used. It is not necessary to know the full name, only a part is enough. Just with the use of the *getDevicebyName* function an object can be created as shown in listing
refcode:devman. This is probably the easiest way to get the devices and check out the Bluetooth capabilities of the used *Andoid* device.

Listing 5.2: Find the right devices with the *DeviceManager*

```
1  DeviceManager devman = new DeviceManager();
2
3  BluetoothDevice ximu = devman.getDevicebyName("Name of the xIMU");
4  BluetoothDevice ioio = devman.getDevicebyName("IOIO Board");
```

To be able to work with the found devices an object of the class *MotionManager* is needed. The constructor needs the two devices as parameters. When the object is created successfully, a connection could be established with the *connect* function. At this point all incoming information are processed by the framework and are ready for use. If necessary, as in

this example application, the *listener* for all events have to be set. This and the call of the constructor are shown in listing 5.3.

Listing 5.3: Create a *MotonManager* object with the previous choosen devices

```
1 final FullscreenActivity thisActivity = this;
2
3 MotionManager manager = new MotionManager(ioio, ximu);
4
5 manager.setListener(thisActivity);
```

After the registration, touch events and orientation data can be received and processed in the functions shown in listing 5.1. Without the implementation of the *MotionListener* its also possible to access all data over the corresponding *get-functions*. To send and receive *JSON-Messages* over WLAN it is necessary to use the *LeviTabLib* library. This library was previously used in the *Beyond Surface* [4]. The whole range of functions and a complete documentation of the functions can be found in the projects documentation. To establish a communication its necessary to create a *MyHTTPRequest* object. The parameters to create this objects are the target IP and the target port 5.4. An alternative is to set or change this parameter via *setter-functions* later.

Listing 5.4: *MyHTTPRequest* object with given IP and Port.

```
1 MyHTTPRequest myRequestForHostX = new MyHTTPRequest(127.0.0.1,80);
```

After that, on a suitable spot a new *JSON* object of a selectable type is needed. Now its possible to send it with the help of the *MyHTTPRequest* object, shown in listing 5.5.

Listing 5.5: Generation *JSON*.Object and sending it via a *MyHTTPRequest*-Object.

```
1 JsonObject myJSONData = new JsonObject(Type.data, "JSON-Data");
2 myRequestForHostX.execute(myJSONData);
```

To receive responses or requests an process them the *StartActivities* listener of the *LeviTabLib* has to be implemented. 5.6.

Listing 5.6: Listener for incomming HTTP-Requests or Responses.

```
1 public class FullscreenActivity extends Activity
```

```
2           implements StartActivities {
3           @Override
4           public void startString(final String string) {
5                   this.runOnUiThread(new Runnable() {
6
7                           @Override
8                           public void run() {
9                            //if you want to manipulate the UI
10                           //just out of this thread
11                          }
12  }
```

6 LIMITATIONS

During the project, some problems have occurred. Because of the inaccurate tablet sensors an external IMU had to be used. The problem with the available *x-IMU* was the framework written in *C++*. Because of that it could not be used out of the box and had to be rewritten in Java. Most of the code was translated fast, but some parts have to be written entirely new. Another problem faced in the project was the hardly documented byte protocol which was send form the *x-IMU* to the original *C++* framework. So it costs a lot of time to find out which byte order defines which packet and in which order the values were transmitted. Despite all these problems the *x-IMU* delivered very accurate orientation data. With the option to tare the *x-IMU* in its current position, by sending command packets, it was quite comfortable to use. The biggest problem that still exists it the mechanical inaccuracy of the gears. Because of the fact that there is too much room to move between the gearwheels it is not possible to get the desired accuracy. The best results could be achieved by turning off of the motors before they reached the desired angle. In this point the prototype should be improved. But altogether this prototype and the developed framework are quite good to use. It opened a new way to explore volumetric data and show up new opportunities to interact with hovering tablets.

Bibliography

[1] Kinetic Sculpture for the BMW Museum. http://vimeo.com/8554267.

[2] T. Alrøe, J. Grann, E. Grönvall, M. G. Petersen, and J. L. Rasmussen. Aerial tunes: exploring interaction qualities of mid-air displays. In *Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design*, NordiCHI '12, pages 514–523, New York, NY, USA, 2012. ACM.

[3] J. Lee, R. Post, and H. Ishii. ZeroN: mid-air tangible interaction enabled by computer controlled magnetic levitation. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 327–336, New York, NY, USA, 2011. ACM.

[4] F. Maier, F. Schwab, and S. Herrdum. Beyond surface - hci project of ulm university. Januar 2014.

[5] M. Marshall, T. Carter, J. Alexander, and S. Subramanian. Ultra-tangibles: creating movable tangible objects on interactive tables. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, CHI '12, pages 2185–2188, New York, NY, USA, 2012. ACM.

[6] G. Pangaro, D. Maynes-Aminzade, and H. Ishii. The actuated workbench: computer-controlled actuation in tabletop tangible interfaces. In *Proceedings of the 15th annual ACM symposium on User interface software and technology*, UIST '02, pages 181–190, New York, NY, USA, 2002. ACM.

[7] Sparkfun. Software, firmware and hardware of the ioio - i/o for android. https://github.com/ytai/ioio, Mai 2014.

[8] M. Spindler. Spatially aware tangible display interaction in a tabletop environment. In *Proceedings of the 2012 ACM international conference on Interactive tabletops and surfaces*, ITS '12, pages 277–282, New York, NY, USA, 2012. ACM.

[9] M. Spindler, W. Büschel, and R. Dachselt. Use your head: tangible windows for 3D information spaces in a tabletop environment. In *Proceedings of the 2012 ACM international conference on Interactive tabletops and surfaces*, ITS '12, pages 245–254, New York, NY, USA, 2012. ACM.

[10] M. Spindler, C. Tominski, H. Schumann, and R. Dachselt. Tangible views for information visualization. In *ACM International Conference on Interactive Tabletops and Surfaces*, ITS '10, pages 157–166, New York, NY, USA, 2010. ACM.

[11] M. Tsang, G. W. Fitzmzurice, G. Kurtenbach, A. Khan, and B. Buxton. Boom chameleon: simultaneous capture of 3D viewpoint, voice and gesture annotations on a spatially-aware display. *ACM Trans. Graph.*, 22(3):698–698, July 2003.

[12] xioTechnologies. xiotechnologies on github. https://github.com/xioTechnologies, Mai 2014.