LEIBNIZ UNIVERSITÄT HANNOVER

Faculty of Electrical Engineering and Computer Science
Human-Computer Interaction Group

CASUAL INTERACTION WITH A SMARTWATCH

A Thesis presented for the degree of Master of Science

by
SVEN RÖTTERING
July 2016

First Examiner	:	Prof. Dr. Michael Rohs
Second Examiner	:	Prof. Franz-Erich Wolter
Supervisor	:	M.Sc. Henning Pohl

EIDESSTATTLICHE ERKLÄRUNG

Hiermit versichere ich, die vorliegende Arbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die wörtlich oder inhaltlich aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Hannover, July 2016

Sven Röttering

ABSTRACT

This thesis investigates which and how user interfaces allowing for *Casual Interaction* should be implemented. A mobile music player as a representative application is implemented on an android handheld device, which is fully controllable by an android smartwatch. Interactions between user and music player can happen via touch input, speech commands or arm gestures. Subsequently a case study investigates which interaction techniques are preferred by users depending on task and context. For this purpose the participants are put in 6 different scenarios in which they have to follow predefined actions regarding the music player.

ZUSAMMENFASSUNG

Diese Arbeit untersucht welche und auf welche Weise Nutzerschnittstellen, die *Casual Interaction* nutzen, implementiert werden sollten. Als Beispielanwendung wird ein mobiler Musikplayer auf einem Android Smartphone implementiert, der vollständig per Android Smartwatch gesteuert werden kann. Touch-Eingabe, Sprachbefehle und Armgesten stehen dem Benutzer dabei als Interaktionsmöglichkeiten zur Verfügung. In einer Studie wird anschließend untersucht, welche Interaktionsmöglichkeiten, abhängig von Aufgabe und Kontext, von den Benutzern bevorzugt werden. Dazu werden die Teilnehmer in 6 verschiedene alltägliche Situationen versetzt, in denen sie Vorgaben erhalten, welche Funktionen des Musikplayer sie steuern sollen.

CONTENTS

1 MOTIVATION

Electronic devices connected to the internet play a big role in almost everyone's life. According to a survey[1] two thirds of the global population have access to the internet and almost every third human owns a smartphone in 2016. People not being bound to stationary computers anymore has numerous advantages and opens up for manifold possibilites of applications. However most applications are exclusively touch based and hence draw a user's attention from her environment to the screen in a large extent. Depending on the context, this inobservance c various levels of adverse impact – one merely could have to reread a passage of a book or in fact cause a serious car accident with lethal consequences. Furthermore, a user is not always able to interact with a handheld device as intended due to physical or mental barriers.

Casual Interaction addresses this issue by introducing additional input techniques and interactions which offer the required flexibility in terms of control over an application, such that the user can limit the required amount of focus to a minimum [9]. The H-metaphor [7] expresses this by looking at a horse rider. Whenever high amount of control is needed, the rider can steer the horse precisely with the reins. However, riding straightly requires nearly no effort from the rider.

The music player implemented in this thesis offers a speech and armgesture interface for input in addition to the typical touch based user interface. This enables the user to leave her handheld device in the pocket and interact with it even when her hands are not available. A typical daily life scenario involving the music player might be the following:

> Programmer Bob is a passionate music listener. At work he starts the day with his favorite playlist with shuffle mode enabled to boost his coding efficiency. Occasionally some songs hit his ears that he heard enough of in the last days so he performs a short arm swing to the right to skip to the next song without loosing track of his current programming task.
>
> After work he rides his bike home, still listening to music, to have a physical compensation for sitting in a chair half the day. In order to hear other traffic participants and emergency sirens he decides to turn the volume of the music down a bit by raising his smartwatch

> to his mouth and saying *"volume down, please"*. Arriving at home he switches from listening with headphones to his brand new sound system that is spread around his entire house. During dinner preparations he gets a phone call from a friend. His phone resides one the table, however Bob's hands are still dirty from cooking so he tells his watch to pause the music and answer the call. They arrange on having dinner together today at Bob's place. It strikes Bob that his music library is not prepared for such a dinner but he is able to delegate the creation of a playlist to his music player by simply specifying suitable audio features. Not having to bother about finding the right music enables Bob to finish cooking just before his guest arrives.

This fictive scenario reveals the necessity of alternative input techniques to perform casual interactions in situations where users are physically or mentally obstructed. In order to decide which additional input techniques an application should provide and how the corresponding user interfaces should be designed one has to consider the possibilities these techniques offer as well as how users approach them. The last aspect particularly depends on the user's preferences and perception performing the interactions. On the one hand, the perceived amount of control for a particular level of engagement is important. *Is she able to achieve the desired reaction of the application or does it feel like the application has developed it's own life?*

On the other hand, users often times get influenced by how interactions appear to the environment. *Can i perform an arm gesture right now or will people stare at me if i suddenly wave my arm through the air?* These and related concerns need to be kept in mind in order to be able to develop useful and effective casual user interfaces.

However, devices require certain hardware features to enable such interactions in the first place. First, it should stay where it is needed without encumbering the user. Typical remotes or smartphones occupy at least one hand for every interaction they offer. Since this is not beneficial a wearable device is needed that is attached to the body without obstructing everyday activities. Second, the device should offer touch-free interaction. This can be realised by adding movement sensors (e.g. accelerometer or gyroscope) and a microphone.[5]

This thesis builds on the previous work of Karoline Busse [5] who developed a wrist-worn silicone bracelet intended for usage with lighting systems. The bracelet is missing on some important components, though, to gain more potential, namely a microphone and a display. Smartwatches basically offer the most important hardware components needed for creating a comfortable and enjoyable casual interaction experience thus being a perfect device for the further studies of this thesis.

A music player is chosen as a representative everyday application. The music player is connected to a private Spotify account via the Spotify Android Software Development Kit (SDK) [1] which serves the music library. Touch, speech and arm gesture input for player control realize different levels of engagement.

Chapter 2 first outlines the related work. Chapter 3 then gives an insight into the implementation of the music player's core features. Subsequently the user study design is addressed in chapter **??** and the resultant data is evaluated in chapter **??**.

Finally, chapter **??** discusses the findings, draws a conclusion and provides ideas for future improvements to casual interactions.

1 https://developer.spotify.com/technologies/spotify-android-sdk/

2 RELATED WORK

Casual interaction has become a big research topic in human-computer interaction (HCI) nowadays.

Pohl and Murray-Smith [9] have characterised the term casual interaction in contrast to focused interaction and described the *focused-casual continuum*, which is a control-theoretic framework that characterizes input techniques in regard to how much flexibility, in terms of thinking and effort, they allow a user to invest into interactions. They showed in a user study that users adjust their level of engagement to the task's complexity.

On this basis, [5] constructed a wrist worn silicone bracelet. When worn, a user could casually interact with a light source. Simple actions like turning the light on and off up to picking individual colors with a capacitive touch stripe. Accelerometer based gestures could be used to activate previously defined and memorized light settings. Despite being highly accessible on the wrist, a user would still have to utilize the hand without the bracelet to activate it's features making interactions rather impractical in certain situations.

Another approach places a depth camera for capturing hand gestures on the user's foot pointing upwards [2]. This allows for discreet interactions thus neglecting concerns of social acceptability of performing gestures as they found out. In a lab study they compared physical and mental demand, user preferences and demonstrated a 94-99% recognition rate.

An alternative input technique is shown in [10]. They introduce around-device devices. Input is received by observing position and rotation as well as arrangement or absence of the around-device devices. To capture this information they propose placing a smartphone equipped with a depth camera nearby. In contrast to the aforementioned approaches, this technique is limited to stationary contexts automatically excluding any in-motion-situations.

Furthermore, casual interaction was applied to mobile music retrieval by [4]. They investigated the listening habits of 95 last.fm[1] users and divided them into three groups. The first group consists of the engaged listeners who invest high initial engagement by e.g. selecting a specific album and afterwards only make quick and decisive interventions. The second group consists of the casual users who invest little effort in interventions at any time. The third group is a mixture of the first two groups where music listening behaviour highly depends on the context. Based on these groups they added a seman-

1 Internet radio station: `www.last.fm`

tic zooming view of linear music space to a already given music retrieval interface. Zooming in on the view enables the user to make more specific music selections. A recommender system additionally infers other relevant music depending on the input specificity.

In the scope of interactions with smart home appliances [8] conducted a series of user study on gestural input for devices found in an average living room – namely blinds, lamps, tv, Electronic Program Guide (EPG), video recorder and answering machine. In the first study they tried to determine a gesture vocabluray. Therefor they observed eigtheen participants seated on a sofa in fully functional living room with the above mentioned devices. The participants were asked to perform a gesture, they would deem appropriate, for every action or referent as [12] refer to. In a second study 22 new participants should then map the gestures from the vocabulary back to the referents. The last study was performed by 10 participants to study the memorability. In a training session every participant performed every gesture five times and then rated the suitability. Finally a slide show displayed every referent for 5 seconds in a random order. If the participant could not perform the gesture in this time, the correct gesture was shown again and the referent was added to the end of the slide show again. Overall their results showed, that simple and short physically or symbolic inspired gestures were rated most suitable and appeared to be most memorizable.

In Addition, [6] conducted two related user studies. 28 participants were asked to propose gestures for referents similar to the above mentioned. A month later, the same participants had to choose the most suitable gesture for each gesture group (i.e. for each referent) which also included their own derived gesture. It turned out, that 65% of the top gestures from the first experiment were not the most chosen gestures in the second experiment. For some of the most agreed gestures in the second experiment, e.g. rubbing one's shoulders for turning off air conditioning, the frequency was only below 10% in the first experiment. Their results show that considering only the most frequent matching user derived gestures can not automatically be considered to be most suitable.

Casual interaction through speech input is yet to be explored. Some research, however, was inquired in the field of smart homes. For example, [3] prototyped a cooking assistant that was installed in their Ambient Living Testbed. Users could interact with the assistant either via touchscreen, mouse and keyboard or via speech input. The latter came in handy while being physically distracted as they were searching for ingredients or cutting vegetables. The findings from the user study based on the cooking assistant revealed that users prefer the availability of mutliple modalities as the possibility to fall back to touch oder mouse input provides an idea safety against failures of the voice recognition. Furthermore, they state a higher acceptance of command-based speech interactions instead of entering whole sentences, as short commands are easier to learn. For more complex commands, users tend to ask the system for help.

3 IMPLEMENTATION

This chapter gives an overview of the implementation aspects of the smartwatch controlled mobile music player. It explains the application structure, the chosen interaction techniques and the mobile-wear communication system.

The essential music library is provided by a private Spotify[1] premium account. The prototype is divided into two separate applications, i.e. mobile (handheld) and wear (smartwatch) application. Users are able to access all features of the mobile application via the smartwatch application. In order to make use of the focused-casual continuum [9], three interaction techniques, differing in the amount of control granted, are available for the user:

- Touch
- Speech
- Gesture

Each technique can be used for the simpler actions of the music player such as play and pause, skip to previous or next song and changing the volume. However, for the more complex interactions, e.g. choosing a playlist, only the touch and speech interaction methods suffice. Gesture and speech input can be performed casually while not even looking at the device (the smartwatch). Section 3.3 describes the functionality and the power of the different methods.

3.1 HARDWARE REQUIREMENTS

Both applications are implemented for the android platform. The handheld device has no further hardware or software requirements other than supporting the android wearable Application Programming Interface (API) so a Samsung Galaxy S6 is chosen. The smartwatch application certainly requires the device to be equipped with a acceleration sensor and a microphone. A moto360 from Motorola is well suited for this. Both devices need to support Bluetooth, too.

3.2 APPLICATION STRUCTURE

The application is divided into two dedicated android applications, one for the mobile device and one for the smartwatch. All of the

1 Music streaming service: `www.spotify.com`

music players functionality is implemented in the mobile application where the Spotify Android SDK and the Spotify Web API for Android[2] is used. Together they allow for every possible interaction with Spotify such as streaming music, creating playlists, searching for artists or tracks, etc. The wearable application provides an additional GUI which contains the same information as the mobile application's GUI with less overhead (e.g. images). It also is responsible for hosting the touch, speech and gesture input techniques. Figure 1 shows each application's different components which are each separated by java packages.

MOBILE APPLICATION The *view* package contains the MainActivity of the application. It is responsible for both maintaining the GUI elements (see section 3.3) and turning issued commands into music player actions which are then forwarded to the SpotifyManager class. The *speechcontrol* package contains the tool which interprets a speech command sent from the wearable application. See section 3.3.2 for a detailed description about speech commands. The *core* package contains both the CommunicationManager class that handles messaging between wearable and mobile application and the SpotifyManager class which ties the Spotify Android SDK and the Spotify Web API for Android together. On the application start it downloads information about the music items that belong to the logged in Spotify account and serves it to the GUI. Audio features are downloaded for each track in the user's library as well. They are stored in the AudioFeatureDatabase class in form of a Hashmap with the track's id as key and a list of the audio features as the value. The AudioFeatureDatabase class is also responsible for searching tracks from the library with a specified audio feature value (see 3.3.2 for what this is useful and how it works). Audio features are also saved to the mobile phones disk by the SpotifyCache class to reduce the amount of data that has to be downloaded on the application start.

WEARABLE APPLICATION The wearable application's structure is similar to the mobile application. Classes responsible for maintaining the GUI elements are located in the *view* package. The *wiigee* package contains a pre-built event driven library for acceleration based gesture recognition (see section 3.3.3 for further details). The *core* package contains the CommunicationManager that again is responsible for messaging between wearable and mobile application. Also the system for recording speech and transforming it into text is located in the *core* package. As soon as voice recognition is enabled (see section 3.3.2) the VoiceRecognitionListener class receives the recognized text from android's Speech Recognizer and waits until the final key-

2 `https://github.com/kaaes/spotify-web-api-android`

word is included. The text is then sent to the mobile application to be interpreted.

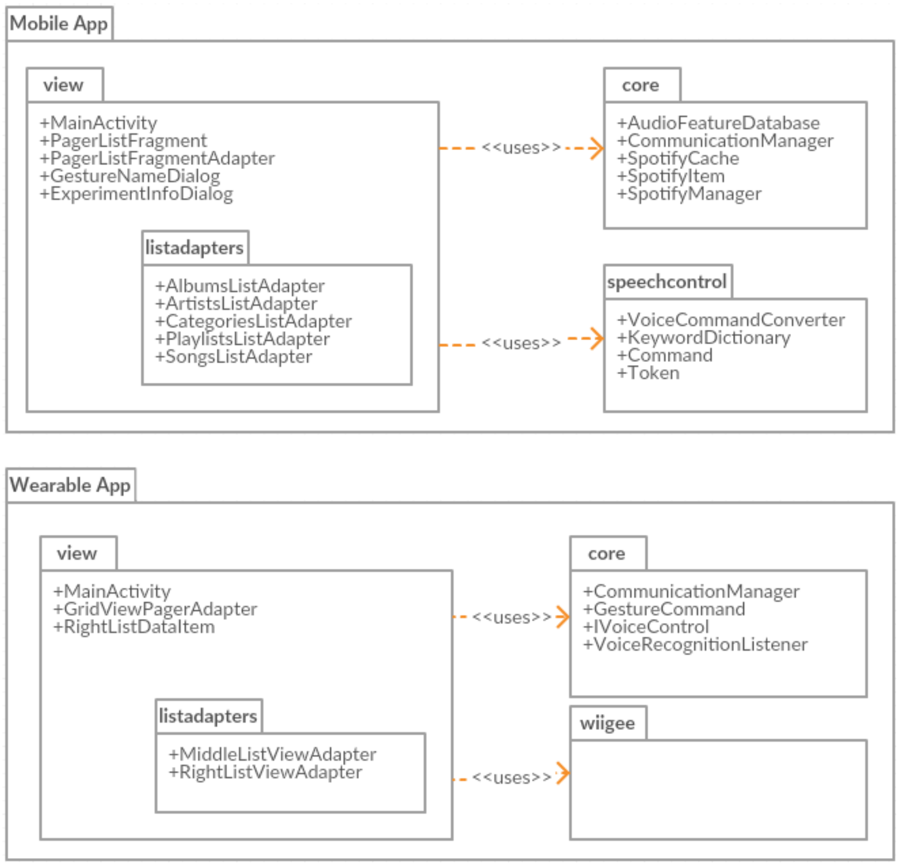


Figure 1: Package diagram of the mobile (top) and wearable (bottom) application.

3.3 USER INTERFACE

The GUI from the mobile application only differs from the wear GUI in terms of appearance. In terms of control over the music player, both versions offer the same possibilities and show the same information (when connected) at any given time. The wear application, however, additionally introduces speech and gesture interactions which the mobile application is not capable of.

It is noteworthy that the Spotify SDK offers a lot more functionality than implemented, such as creating own playlists or searching the entire Spotify music library for keywords. The implemented music player, however, just offers the basic playback functionalities in order to keep the application simple and to not overload the experiment participants with information since they have to remember how to control the music player.

3.3.1 *Touch Input - GUI*

Categories are Spotify's extended version of genres

Both GUIs are designed to be simple and straightforward. The mobile version mainly consists of two areas. Figure 2 depicts a screenshot of this layout. A control panel is situated at the bottom of the screen. Above this is a left-right scrollable pager containing different kinds of lists. The scrollable list pager contains five lists, one for each music arrangement which are playlists, songs, albums, artists and categories. The control panel located beneath the list pager contains four control

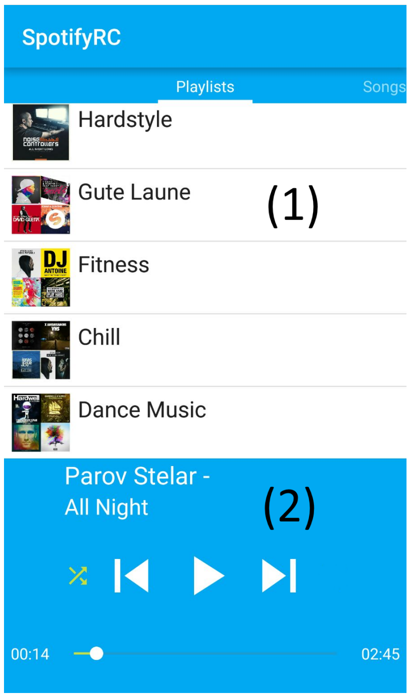
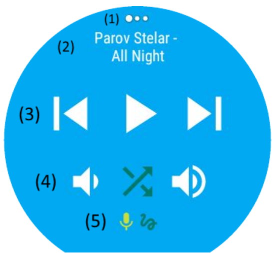


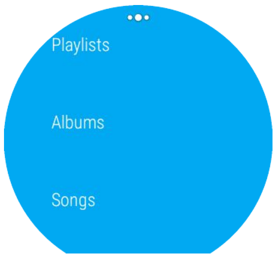
Figure 2: GUI of the mobile application. (1) shows the left-right scrollable list pager with indicators which list is shown at the top. (2) shows the player control panel in blue.

buttons such as a shuffle button including on-off indicator, a skip to previous song button, a play-pause button and a skip to next song button. Unlike the wear application, the mobile version spares buttons for controlling the volume, because most devices own hardware buttons for this purpose. Information on the name of the current song and artist can be found above these buttons. The track's progress and total duration can be found at the very bottom of the screen.

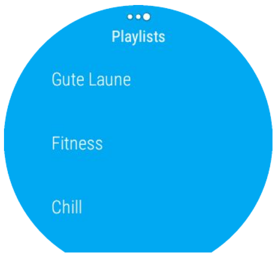
However, the intention of the wear application is, that the user does not need to bother reaching his mobile phone. For this reason, the wear application's GUI offers the same amount of touch control over the music player. Figure 3 demonstrates the layout of the wear application. It consists of a GridViewPager with three horizontal pages.



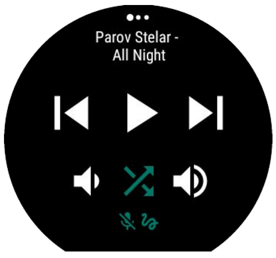
(a) control panel in interactive mode



(b) list of music arrangements



(c) list of playlists



(d) control panel in ambient mode

Figure 3: The GUI of the wear application consists of a GridViewPager containing three horizontal pages. A dot indicator at the top of the screen shows the current page. 3d depicts the controls page in ambient mode, i.e. battery saving mode, while 3a is the same screen in interactive mode. Swiping right brings up the list of music arrangements 3b. Selecting e.g. playlists switches to the third page 3c showing a list with all playlists.

Figure 3a shows the main control page which is divided into five rows (1) - (5). A dot indicator showing the current page with a bigger dot is located at the top of the screen in row (1). Row (2) contains the current artist and track that is playing. Row (3) and (4) contain the control buttons, i.e. in (3) the buttons for play previous song, toggle play and pause, play next song and in (4) the buttons for decrease volume, toggle shuffle (grey = off, green = on) and increase volume are located. Row (5) contains two icons which indicate whether speech and gesture recognition are active (green) or inactive (grey).

Figure 3b shows the page in the middle which is reached by swiping over the screen from right to left. The page contains a vertical scrollable list of the five music arrangements, namely playlists, songs, albums, artists and categories. Selecting a list entry scrolls to the right (the third) page which always adapts its list showing the respective items in the selected arrangement.

The moto360 smartwatch is equipped with a battery saving mode called ambient mode. In this mode cpu processing is reduced and it is recommended[3] to reduce the User Interface (UI) layout to a black background and white text color. Updates to the UI should then happen on a several seconds up to a minute basis. Applications that run in both ambient and interactive mode are called *always-on apps*. The transition from interactive to ambient mode happens either automatically after a short period of user inactivity or can be forced by covering the screen. Leaving ambient mode happens either by touching the screen or by bringing up the wrist as in looking at the time. Figure 3d depicts the control page of the wear application running in ambient mode with a black background and white text and icon color.

3.3.2 *Speech Control*

The wear application offers the possibility to control the music player via speech commands. In order to enter a speech command, the application must be in interactive mode which activates the continuous speech recognition (indicated by the green microphone seen in 3a at the bottom of the screen). The user can then talk to the watch and issue a command. Speech data is recorded and converted to text by the android speech recognition service[4]. Since the speech recognition happens continuously input must be recognized easily as a possible valid command. Therefor every command has to end with a special final keyword, the word *"bitte"*. As soon as the wear application detects this keyword, the converted text is send to the mobile application where it gets parsed, transformed into a command and finally executed. Figure 4 illustrates this procedure with an activity diagram. The wear-mobile communication process is described in more detail in section 3.4.

"Bitte" is the german word for "please"

Speech commands for this music player can be divided into three groups based on their complexity. The first group contains the simplest commands that just consist of one keyword. Commands in the second group start with a keyword and are followd by additional words, until the final keyword occurs, that are handled as the input parameters. Commands in the third group are built by chaining different keywords together forming a simple sentence.

Table 1 contains the respective keywords for commands in the first two groups. The first six simple keywords each represent a command by itself and thus belong to group 1. The *play* command is the only command in group 2 meaning it requires a parameter to work as intended.

3 `https://developer.android.com/training/wearables/apps/always-on.html`
4 `https://developer.android.com/reference/android/speech/SpeechRecognizer.html`

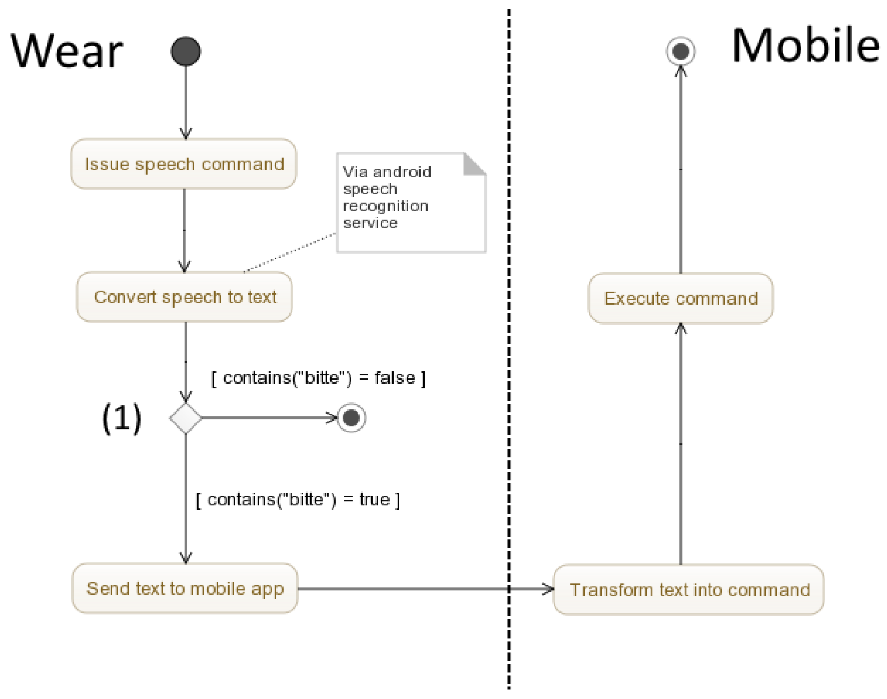

Wear
Mobile
Issue speech command
Via android speech recognition service
Convert speech to text
[contains("bitte") = false]
(1)
[contains("bitte") = true]
Send text to mobile app
Transform text into command
Execute command

Figure 4: Activity diagram of the speech interaction procedure. Actions to the left of the dashed line happen in the wear application and actions to the right of the dashed line happen in the mobile application. At stage (1) a preliminary decision is made whether the converted text resembles a command or not.

Commands in the third group, which are built by chaining the keywords shown in table 2 together, offer a special kind of control over the music player. They allow for queuing and playing tracks that differ in terms of an audio feature from the current playing track.

AUDIO FEATURE: Spotify runs a suite of audio analysis algorithms on every track in their catalog. These extract about a dozen high-level acoustic attributes from the audio. Some are well-known musical features, like tempo and loudness. Others are more specialized, like valence and energy.

The four most intuitive audio features are supported by the music player to be used with speech commands, which are[5]:

TEMPO: The estimated tempo of a track measured in Beats per Minute (BPM). It derives directly from the average beat duration.

ENERGY: A measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale.

5 A complete list of audio features and their meaning can be found at: `https://developer.spotify.com/web-api/get-several-audio-features/`

SPEECH COMMAND KEYWORDS	ACTIONS IN MUSIC PLAYER
pause	pause music playback
weiter	resume music playback
nächstes	skips to next song
zurück	skips to previous song
lauter	increases the volume
leiser	decreases the volume
play <music element>	load and play an element from the music library

Table 1: Speech commands from the first two groups. The play command is the only command accepting parameters. All other commands in this table belong to group 1. To issue a command, simply activate speech recognition by entering the watches interactive mode and say the keyword followed by the final keyword "bitte".

LOUDNESS: The overall loudness of a track in decibles (dB). Loudness values are averaged across the entire track. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values range between -60 and 0 db.

VALENCE: A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive, while tracks with low valence sound more negative.

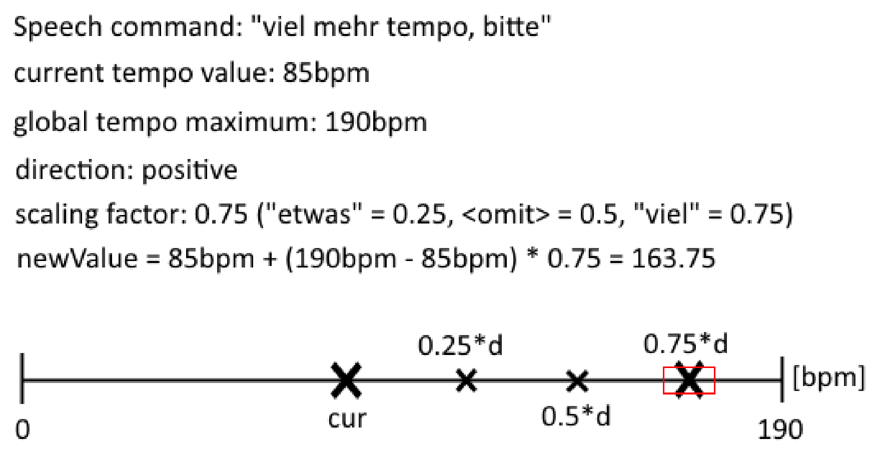
SCALING	DIRECTION	AUDIO FEATURE
etwas	mehr	Tempo
viel	weniger	Energie
		Lautstärke
		Stimmung

Table 2: Building blocks (keywords) for commands from the third group. Choose a keyword from each column, chain them to a sentence and attach the final keyword "bitte" at the end. The scaling keyword can be omitted.

Changing tracks based on the current track's audio features enables the user to specify what kind of music she wants to listen to in a situation, without deciding on the exact tracks to be played. The pseudo code in listing 1 is responsible for searching the corresponding tracks in the music library. The new value for the specified audio feature is calculated in line 2 respectively 4 by first evaluating the direction keyword which determines whether to increase or decrease the current value. The scale keyword then determines the scaling factor which indicates by how much the current value is changed. Therefor the scaling factor is multiplied by the distance between the current value and either the global minimum or maximum value for the given audio feature depending on the direction keyword. Afterwards, tracks in range of the new calculated value, determined by a tolerance radius, are searched (line 7 and forth). The tolerance radius is increased by a fix amount as long as no tracks are found. Figure 5 explains this mechanism with example values.

Listing 1: Pseudo code for calculating the new audio feature value and looking up respective tracks with a tolerance radius from the music library

```
1  if(direction == negative) {
2          newValue = cur + (cur - min) * -scale;
3  } else {
4          newValue = cur + (max - cur) * scale;
5  }
6
7  while(noTracksFound) {
8          lookupTracksWithValueAndTolerance(newValue, tolerance);
9          increase(tolerance);
10 }
```



Speech command: "viel mehr tempo, bitte"

current tempo value: 85bpm

global tempo maximum: 190bpm

direction: positive

scaling factor: 0.75 ("etwas" = 0.25, <omit> = 0.5, "viel" = 0.75)

newValue = 85bpm + (190bpm - 85bpm) * 0.75 = 163.75

Figure 5: Example calculation for changing tracks based on the audio's tempo. The big X's show the current and the new value, while the smaller ones depict values with a smaller scaling factor. The red rectangle around the new value illustrates the tolerance radius in which new tracks are searched.

After sending a possible speech command to the mobile application, the text needs to be converted to an applicable command. For this, a conversion algorithm first searches the text for command keywords from the first two speech command groups (see table 1) and creates a token of a respective type for it when found. Tokens that belong to the simple commands in the first group are directly mapped to their corresponding music player command and then executed.

Item names of songs and albums are a combination of their name and the artist

In case the text contains a *play* keyword which is followed by at least one word, an additional token is created for it, being the parameter of the command. The algorithm then tries to find the desired music library item by comparing the item name with the parameter text. Because item names often contain additional artifacts, such as *Radio Edit* or *Original Mix*, and the speech to text conversion is not always accurate, comparison is not based on equality but rather on similarity. The pseudo code in listing 2 finds the item name with the highest similarity to the parameter text based on the *Levensthein distance*[6]. An item name which equals the parameter text can be considered as most similar. However, in case the item name is longer than the command's parameter, the foreach loop starting in line 7 computes the minimum distance for every substring of the item name with the same length of the parameter. In the end, the item with the minimum distance which is equal to the highest similarity to the parameter is returned and played.

Listing 2: Calculating parameter and music item name similarity

```
 1 var mostSimilarItem = null;
 2 foreach(item in library) {
 3   if(parameter == item.name) {
 4     return item; // distance is already 0, so we found the item
 5   }
 6   var minDistance = Integer.MAX;
 7   foreach(substring s of item.name with s.length == parameter.
        length) { // if the parameter is longer than the item name,
         only one iteration happens
 8     distance = Levensthein(parameter, s);
 9     if(distance < minDistance) { // less distance equals higher
          similarity
10       minDistance = distance;
11       mostSimilarItem = item;
12     }
13   }
14 }
15 return mostSimilarItem;
```

If no keywords for group 1 or 2 are found, the conversion algorithm checks for keywords belonging to the third group and creates a token for each one. The text is a valid command if at least a token for the direction and the audio feature is present. A scaling keyword can be omitted and the order of the tokens is not considered. The tokens are then used to directly set the parameters of the formula for calculating the new value for the specified audio feature (see listing 1).

6 String metric for measuring the number of required edits in one string in order to match the other.

The following commands are examples from every group executed on the music library from a private spotify account:

1. "NÄCHSTES, BITTE": skips to the next song in the queue
2. "PLAY HARDWELL, BITTE": enqueues and plays all tracks by Hardwell from the library
3. "VIEL MEHR TEMPO, BITTE": plays tracks from the library with significantly increased tempo compared to the current

3.3.3 *Gesture Control*

The third interaction technique supported by the implemented music player involves acceleration based gesture recognition. With this the user is able to issue commands to the player without looking at or touching the screen with the other hand. Gesture commands on the contrary are not as powerful as touch input or speech commands. They only offer control for pause and resume, skip to next song, back to previous song as well as increase and decrease volume. The other commands that speech input offers would result in gestures too complex to perform and remember for a user.

For this, a pre-built event-driven gesture recognition library called *Wiigee*[7] is used. Wiigee accepts three dimensional acceleration vectors as input for pattern training and recognition. The vectors are sent through a four-level recognition pipeline which is shown in Figure 6. [11] describe the four stages of the pipeline in detail. They observed an average rate of correctly recognized gestures of 90%.

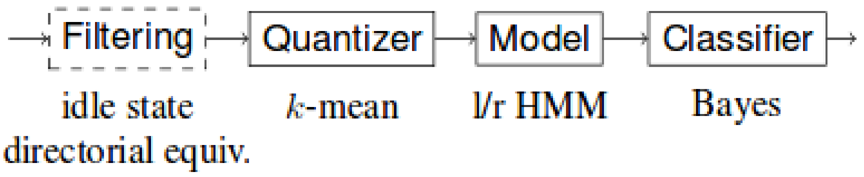
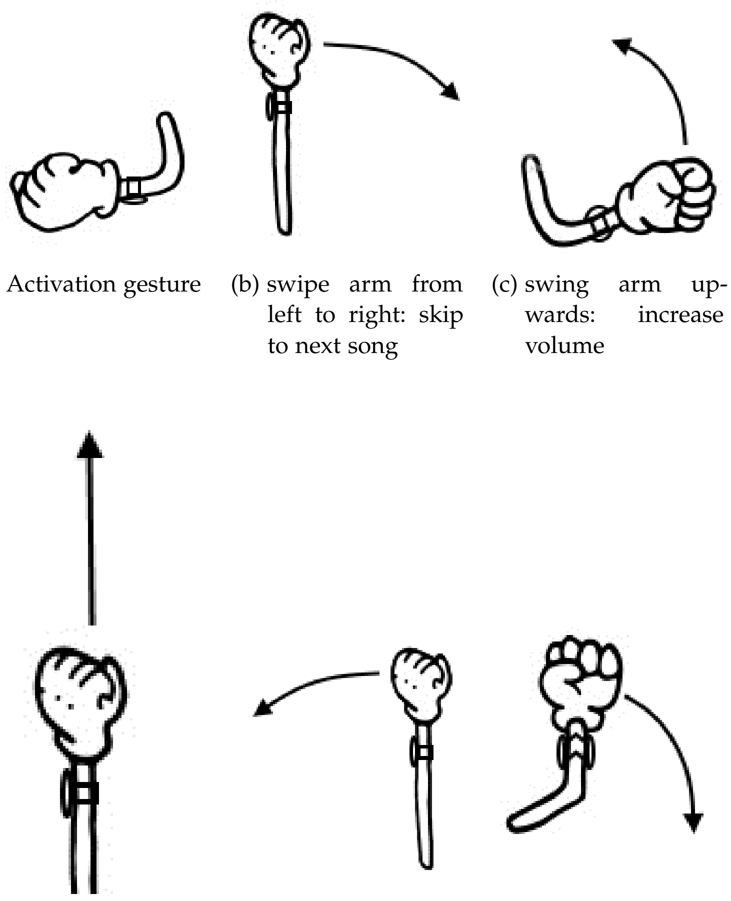


Figure 6: Original from [11]. Four-level recognition pipeline that first filters vector data und clusters it using a k-means algorithm. Subsequently a hidden markov model is fed with the clustered data and classified by a Bayesian classifier.

In order to recognize gestures, they first have to be trained to wiigee. For later use, the data for every gesture can be saved to a file afterwards. [11] claims that a training session with five to ten repetitions is sufficient for Wiigee to learn a new gesture. The gestures for controlling the music player are kept short and simple as [8] suggest. However, to avoid unintentional commands and to save battery life, an activation gesture has to be performed beforehand. Also, Wiigee

7 Official Wiigee site: `www.wiigee.org`



(a) Activation gesture

(b) swipe arm from left to right: skip to next song

(c) swing arm upwards: increase volume

(d) punch forward: play and pause

(e) swipe arm from right to left: back to previous song

(f) swing arm downwards: decrease volume

Figure 7: 7a depicts the activation gesture where the arm is bent and the watch face is turned down until the pitch threshold is reached. When the vibration countdown is over, one of the other five gestures can be performed to issue a command to the music player.

does not support continuous gesture recognition. Figure 7 shows a list of all supported gestures and their meaning including the activation gesture.

Once Wiigee recognized a gesture it notifies the registered listeners sending the recognized gesture id and a confidence score between 0 and 1. In order to filter out false positives, the confidence score has to exceed 0.97 before the gesture is accepted by mapping its id to a music player command.

The activation gesture is performed by rotating the watch with its face down reaching a pitch of at least -70°or lower. For this, the rotation of the watch is constantly measured with android's software-

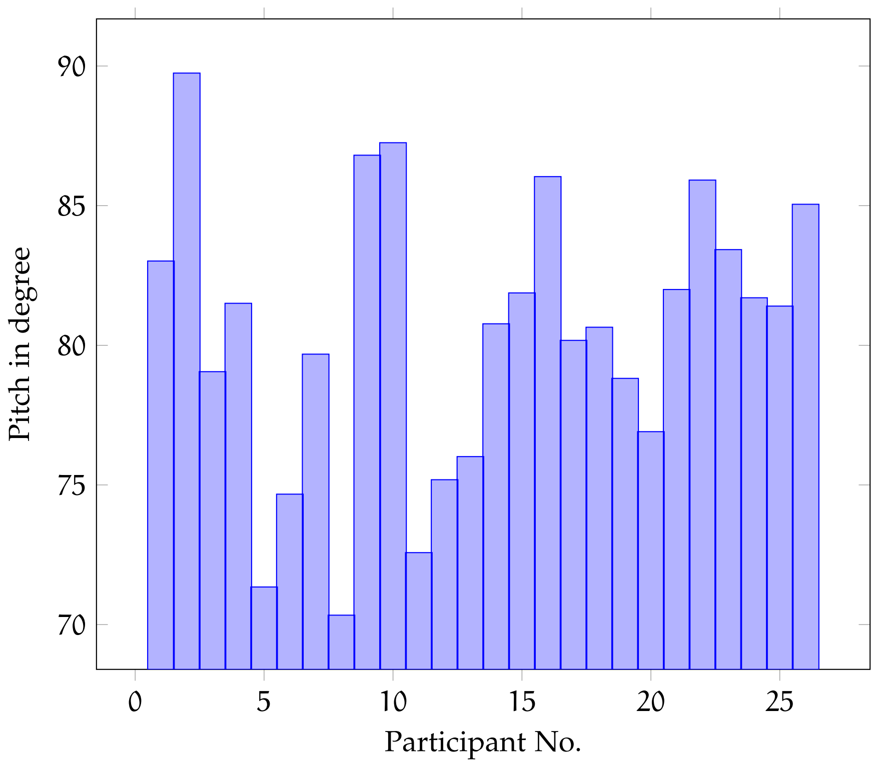


Figure 8: Minimum absolute pitch values for each participant. The minimum absolute pitch value that every participant reached is 70.3

based *Game Rotation Vector Sensor*[8]. When the gesture recognition gets activated, the user receives a sequence of three vibrations as feedback from the watch after which a gesture can be performed. Wiigee's recognition algorithm is sensitive to the device rotation, hence the device orientation during recognition should be approximately the same as it was during the training in order to be recognized correctly.

Since it can not be assumed that users wear their watch in the same position and certainly have different wrist anatomies, a suited pitch value for the activation gesture has to be determined. An informal experiment involving 26 participants has been conducted, where each participant had the task to turn their wrist five times as much as possible without getting uncomfortable. The minimum values from each participant series are plotted in figure 8.

Wiigee is an event-based library where the training and recognition process is started by virtual button presses and completed by virtual button releases. In case of the Wiimote[9], for which wiigee was originally designed, users were able to press real buttons. Gesture recognition on the smartwatch, however, can not be activated by real buttons, therefor training and recognition are activated by a virtual

8 Documentation can be found here: `https://developer.android.com/guide/topics/sensors/sensors_position.html`

9 Wiimote is a remote controller for the Nintendo Wii

button press which is send to wiigee after the vibration countdown from the activation gesture. The virtual button release for finishing the training or recognition process is send automatically to wiigee after 750ms, which is enough time to perform each gesture. Deactivating the recognition process as soon as the device stops moving could not be used in this case, since this event can not be reliably detected.

3.4 MOBILE-WEAR COMMUNICATION

Controlling the music player with the smartwatch requires some communication to happen between the watch and the mobile device. Android provides a solution for this which is called the Wearable Data Layer API[10] that is part of Google Play Services. It offers a communication channel for smartphone and wearable applications. In particular, the *MessageApi* class which is responsibule for sending text messages back and forth suffices for the music player. A message can have different types determined by a string that has to be attached to the message. Table 3 lists all types that can be attached to a message in both the wearable and mobile application as well as their purpose and when they are sent. Figure 9 illustrates in which direction the messages are sent.

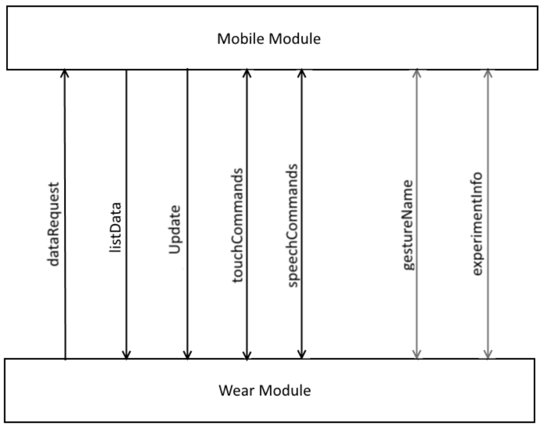


Figure 9: A Message can be of five different types: a data request type, list data type containing information about the music library, an update type containing the current music player state, a touch command type and a speech command type which both are sent from the wearable application and answered by the mobile application afterwards. The two greyed out message types on the right are included for the sake of completeness and are only used for the user study.

10 `https://developer.android.com/training/wearables/data-layer/index.html`

A communication manager class in both applications is responsible for sending and receiving these messsages. When a message is received, the communication manager forwards the message to the application parts concerned which then execute a corresponding action.

MESSAGE TYPE	WHEN SENT	PURPOSE
data request	wear app start.	Indicates the mobile app that music library data is needed.
list data	mobile app start or when a data request was received.	Contains a list of display names for all music items in the library.
update	on music player's state change, i.e. when the track is changed, a play or pause command ocurred or shuffle was toggled.	Contains all information to keep mobile and wear UI synchronized.
touch command	on touch or gesture input on the wear app.	Contains a string representation of the issued command. A gesture id is mapped to the same string as its respective touch command is (see 3.3.3).
spech command	after user completed a speech command	Recorded text is sent to the mobile app to convert it into a command

Table 3: List of the different message types, when they are sent and what their content or purpose is.

4 USER STUDY

The purpose of the music player described in chapter 3 is to examine user behaviour regarding the focused-casual continuum. The idea behind this concept is that users can vary their level of engagement for a certain action while not being bound to interact strictly focused or casual with the device. The three interaction techniques introduced in section 3.3

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

API	Application Programming Interface
BPM	Beats per Minute
EPG	Electronic Program Guide
GUI	Graphical User Interface
HCI	human-computer interaction
SDK	Software Development Kit
UI	User Interface

A APPENDIX TEST

Lorem ipsum at nusquam appellantur his, ut eos erant homero concludaturque. Albucius appellantur deterruisset id eam, vivendum partiendo dissentiet ei ius. Vis melius facilisis ea, sea id convenire referrentur, takimata adolescens ex duo. Ei harum argumentum per. Eam vidit exerci appetere ad, ut vel zzril intellegam interpretaris.

Errem omnium ea per, pro congue populo ornatus cu, ex qui dicant nemore melius. No pri diam iriure euismod. Graecis eleifend appellantur quo id. Id corpora inimicus nam, facer nonummy ne pro, kasd repudiandae ei mei. Mea menandri mediocrem dissentiet cu, ex nominati imperdiet nec, sea odio duis vocent ei. Tempor everti appareat cu ius, ridens audiam an qui, aliquid admodum conceptam ne qui. Vis ea melius nostrum, mel alienum euripidis eu.

A.1 APPENDIX SECTION TEST

Ei choro aeterno antiopam mea, labitur bonorum pri no. His no decore nemore graecis. In eos meis nominavi, liber soluta vim cu. Sea commune suavitate interpretaris eu, vix eu libris efficiantur.

More dummy text.

Nulla fastidii ea ius, exerci suscipit instructior te nam, in ullum postulant quo. Congue quaestio philosophia his at, sea odio autem vulputate ex. Cu usu mucius iisque voluptua. Sit maiorum propriae at, ea cum primis intellegat. Hinc cotidieque reprehendunt eu nec. Autem timeam deleniti usu id, in nec nibh altera.

A.2 ANOTHER APPENDIX SECTION TEST

Equidem detraxit cu nam, vix eu delenit periculis. Eos ut vero constituto, no vidit propriae complectitur sea. Diceret nonummy in has, no qui eligendi recteque consetetur. Mel eu dictas suscipiantur, et sed placerat oporteat. At ipsum electram mei, ad aeque atomorum mea.

LABITUR BONORUM PRI NO	QUE VISTA	HUMAN
fastidii ea ius	germano	demonstratea
suscipit instructior	titulo	personas
quaestio philosophia	facto	demonstrated

Table 4: Autem usu id.

Listing 3: A floating example

```
for i:=maxint to 0 do
begin
{ do nothing }
end;
```

Ei solet nemore consectetuer nam. Ad eam porro impetus, te choro omnes evertitur mel. Molestie conclusionemque vel at, no qui omittam expetenda efficiendi. Eu quo nobis offendit, verterem scriptorem ne vix.

BIBLIOGRAPHY

[1] Smartphone ownership and internet usage continues to climb in emerging economies. `http://www.pewglobal.org/2016/02/22/smartphone-ownership-and-internet-usage-continues-to-climb-in-emerging-economies/`. Accessed: 2016-04-16.

[2] Gilles Bailly, Jörg Müller, Michael Rohs, Daniel Wigdor, and Sven Kratz. Shoesense: a new perspective on gestural interaction and wearable applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1239–1248. ACM, 2012.

[3] Marco Blumendorf, Sebastian Feuerstack, and Sahin Albayrak. Multimodal smart home user interfaces. In *Proc. of IUI4AAL Workshop on IUI*, 2008.

[4] Daniel Boland, Ross McLachlan, and Roderick Murray-Smith. Engaging with mobile music retrieval. In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 484–493. ACM, 2015.

[5] Karoline Busse. Casual interaction with a bracelet. Master's thesis, Leibniz Universität Hannover, 2014.

[6] Eunjung Choi, Sunghyuk Kwon, Donghun Lee, Hojin Lee, and Min K Chung. Can user-derived gesture be considered as the best gesture for a command?: Focusing on the commands for smart home system. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 56, pages 1253–1257. SAGE Publications, 2012.

[7] Frank O Flemisch, Catherine A Adams, Sheila R Conway, Ken H Goodrich, Michael T Palmer, and Paul C Schutte. The h-metaphor as a guideline for vehicle automation and interaction. 2003.

[8] Christine Kühnel, Tilo Westermann, Fabian Hemmert, Sven Kratz, Alexander Müller, and Sebastian Möller. I'm home: Defining and evaluating a gesture set for smart-home control. *International Journal of Human-Computer Studies*, 69(11):693–704, 2011.

[9] Henning Pohl and Roderick Murray-Smith. Focused and casual interactions: allowing users to vary their level of engagement. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2223–2232. ACM, 2013.

[10] Henning Pohl and Michael Rohs. Around-device devices: my coffee mug is a volume dial. In *Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services*, pages 81–90. ACM, 2014.

[11] Thomas Schlömer, Benjamin Poppinga, Niels Henze, and Susanne Boll. Gesture recognition with a wii controller. In *Proceedings of the 2Nd International Conference on Tangible and Embedded Interaction*, TEI '08, pages 11–14, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-004-3. doi: 10.1145/1347390.1347395. URL `http://doi.acm.org/10.1145/1347390.1347395`.

[12] Jacob O Wobbrock, Meredith Ringel Morris, and Andrew D Wilson. User-defined gestures for surface computing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1083–1092. ACM, 2009.