

Mogeste: A Mobile Tool for In-context Motion Gesture Design

1st Author Name

Affiliation

City, Country

e-mail address

2nd Author Name

Affiliation

City, Country

e-mail address

ABSTRACT

Motion gestures can be expressive, fast to access and perform, and facilitated by ubiquitous inertial sensors. However, implementing a gesture recognizer requires substantial programming and pattern recognition expertise. Although several graphical desktop-based tools lower the threshold of development, they do not support ad-hoc development in naturalistic settings. We present Mogeste, a mobile tool for in-context motion gesture design. Mogeste allows interaction designers to create and test motion gestures using inertial sensors in commodity and custom devices. Therefore, our tool encourages development of gestures with common (e.g., hand) as well as atypical (e.g., ear) body parts. Moreover, the data collection, design, and evaluation of envisioned gestural interactions can now occur within the context of its use.

Author Keywords

Design tool; motion gestures; mobility; wearable devices; gesture recognition.

ACM Classification Keywords

H.5.2: User interfaces—*input devices and strategies; prototyping; user-centered design*. D.2.2: Design tools and techniques—*User interfaces*

INTRODUCTION

Gestural input is inherently quick and expressive since a single motion can indicate the operation, the operand, and additional parameters [7]. For instance, a simple turn of the wrist (gesture) quickly towards the eyes (parameters), expresses the wearer's intent to check the time (operation). As a result, the screen of a smart watch (operand) turns on to reveal the time. Motion gestures are particularly useful in situations where visual attention is limited [31], or when brief interactions with wearables are needed [4].

Commodity or custom-built devices that are handheld (e.g.,

Paste the appropriate copyright/license statement here. ACM now supports three different publication options:

- ACM copyright: ACM holds the copyright on the work. This is the historical approach.
- License: The author(s) retain copyright, but ACM receives an exclusive publication license.
- Open Access: The author(s) wish to pay for the work to be open access. The additional fee must be paid to ACM.

This text field is large enough to hold the appropriate release statement assuming it is single-spaced in Times New Roman 8-point font. Please do not change or modify the size of this text box.

Each submission will be assigned a DOI string to be included here.



Figure 1. Mogeste is a mobile tool for designing gestures (a) with wearable/handheld devices (b) in various naturalistic contexts (c).

smartphones), or body worn (e.g., smart watches, Google Glass) readily support motion gestures on touchscreens as well as in air. Despite their ubiquity in interactive devices, the use of inertial sensors for interaction is not as common as touchscreens, particularly among interaction designers. Therefore, we focus on in-air motion gestures using inertial sensors (e.g., accelerometers, gyroscopes, magnetometers). Key challenges include time, effort, and expertise needed in: 1) building a robust recognizer [36]; 2) finding gestures that are distinguishable from everyday motions [5]; 3) mapping gestural input to application functions [18]; and 4) adjusting for the disconnect between the programming environment and usage environment [1].

To mitigate some of these concerns, we developed Mogeste to support rapid, in-context motion gesture design by interaction designers. Mogeste is a mobile tool that facilitates a create-test-analyze workflow [25] and leverages programming by demonstration [12] (see Figure 1). Mogeste extends previous work in desktop authoring tools for sensor-based interactions by providing a mobile, untethered solution. Additionally, easy integration with commodity and custom hardware allows it to support a variety of on-body interactions, not limited to hand gestures. In this paper, we describe the design criteria underlying Mogeste's development, its design, implementation details, and examples of its use.

RELATED WORK

Sensing of Motion Gestures

Most common gestural interfaces can be characterized as touch- [42], pen- [20], tangible- [39], or motion-based [40]. Combinations of these modalities have also been explored (e.g., pen with touch [19], touch with motion [22], touch plus in-air [9,21]). Although researchers have demonstrated use of electric field [10], electromagnetic field [11], capacitive [35], and acoustic [16] sensing for detecting hand gestures, cameras (cf. [30,40,41]) and inertial sensors are more common.

As mentioned before, Mogeste currently utilizes inertial sensors for designing motion gestures because of their ubiquity. Their lightweight, and small form factor allows them to be strapped to a body part to provide direct sensing of its motion. Although they are commonly employed for activity recognition [6], their use for gestural interaction is relatively underexplored [5]. They have been utilized for detecting taps on and around a device [44], coarse motions such as writing in the air [2], or motion states [26] such as walking, running, or driving to infer interaction needs.

Tools for Gesture Design

Several desktop-based design tools inspire Mogeste. Exemplar introduced the design-test-analyze workflow to authoring of sensor-based interactions [18]. Through a direct manipulation interface, it enabled novices to rapidly explore gestural interaction with a variety of sensors. Mogeste extends Exemplar's workflow and hierarchical data model to a mobile solution.

MAGIC [5], a desktop tool for creation, testing, and evaluation of motion-based gestures with a wrist-mounted 3-axis accelerometer, built on Exemplar in two ways. First, it exposed the machine learning (ML) measures of inter- and intra-class comparison scores and confusion matrices in a graphical representation. Second, it introduced a comparison of intended gestures against a repository of everyday hand motions in the analysis stage. Mogeste borrows comparison scores from MAGIC, provides a simpler UI, and utilizes 3-axis accelerometer and 3-axis gyroscope together for supporting rotational degrees of freedom as well.

Crayons [14] allows a designer to assist with development of a vision-based motion gesture recognizer by manually marking regions of interest. In contrast, Mogeste enables a designer to prototype new gestures by handling complex gesture recognition in the background. Quill [27], a CAPpella [13], and Gesture Coder [28] successfully utilize programming by demonstration [12] for designing pen-based gestures, developing context-aware applications, and programming multi-touch gestures, respectively. Mogeste follows these examples to ease adoption by designers who are interested in early-stage motion gesture design but lack expertise in pattern recognition. And it goes beyond to allow collection of examples in an uncontrolled setting.

THE CHALLENGES OF MOTION GESTURE DESIGN: PERSONAL EXPERIENCE AND A CASE STUDY

In a recent unpublished work about building a wrist-worn wearable input device, we established the need for a gesture recognition tool for data acquisition and visualization, classification, and user evaluation. Instinctively we wanted to use commercial tools (e.g., Matlab) or open-source libraries (e.g., Weka [17]) that provide most of the needed functionality, but learning these complex tools takes prolonged effort, and, still, achieving a streamlined workflow with these disparate tools is challenging. Moreover, our requirements kept changing as our understanding of the sensor data and the processing pipeline improved (a challenge also noted by other researchers [33]). Therefore, we engineered a custom solution.

Although our desktop tool ultimately reduced the time for sensor explorations from days to minutes, its development required significant engineering effort. Consequently, we spent very little time on the actual design. Moreover, tethering to a desktop allowed limited testing of interactions outside the lab. Unfortunately, in user interface research, the time and engineering effort it takes to build, test, and document a robust tool is overwhelming, and not typically considered a research contribution [23].

Case Study: Gesture Design Activity with Novices

To investigate the gesture design process by non-experts, we observed an hour-long in-class activity with 20 pairs of undergraduate computer science and graduate industrial design students. The task was to implement a rule-based system to recognize four motion gestures and utilize them to control a pre-coded music player application. Ultimately, all teams were able to demonstrate at least two gestures.

Most teams selected motions along a single axis, which supports Ruiz et al.'s findings [37]. Students often sat while performing gestures or the non-instrumented member had to carry the laptop recording data for mobility. Despite these similarities, student teams pursued two different strategies: gesture first or data first. Irrespective of their approach, several student teams had challenges in setting up their program to visualize and analyze data for threshold comparisons in real-time. We hypothesize that a mobile motion gesture design tool will mitigate these challenges and allow rapid iterations of gesture design.

WHY MOBILE?

Most rapid prototyping tools in the literature are desktop-based tools [5,13,18,25]. *Desktop* (and laptop) development provides a large screen, establishing multiple wired and wireless connections, and, more importantly, leveraging mature development and visualization tools. However, size, and operation in proximity limit design in naturalistic and *ad hoc* settings, particularly for mobile/wearable sensor-based interactions [1].

Mobile platforms afford portability, boast higher penetration and daily usage, and provide simple direct manipulation interfaces [24]. Moreover, experience with

myriad productivity applications shows that, with good UI design, small screens can be used just as easily. Mogeste embraces mobility for in-context gesture design. It also unifies the development and runtime environment.

In *hybrid* solutions, functions are distributed across mobile and desktop platforms, potentially leveraging the best of both worlds. However, as seen in recent examples [28,29], in this mode mobile is just used as a sensor platform.

DESIGN GUIDELINES

As informed by the challenges noted above, the tool should:

Enable in-context design. In-context gesture design refers to the capability to record gesture examples in naturalistic settings, train a recognizer in real-time, and test gestures as commands in an application’s context. In this unique way it supports unexpected moments of creativity.

Promote a rapid iterative workflow. Designers should be able to design and assess new gestures in iterative cycles.

Support off-the-shelf devices and beyond. Interaction designers may use commodity devices to quickly prototype form and interactions of an envisioned device. Additionally, custom devices should be equally easy to setup and use.

Automate the secondary processes. By managing data collection, storage, visualization, and networking, designers are able to focus on the core task of design.

MOGESTE TOOL DESCRIPTION

Workflow

Mogeste’s iterative workflow consists of three main stages: 1) gesture design; 2) gesture testing; and 3) analysis.

Gesture Design and Testing

Designers use Mogeste on a mobile phone platform to create gestures, either one at a time or in groups. For the latter, a designer would first create a gesture group, say “Music Control”, add the names of desired gestures, and then start providing examples for each. Mogeste first reveals a *gesture group list* (Figure 2a) where a user can create new groups, or open and update an existing group. Each group is represented with a name, description, device icon, and a count of gesture classes it contains. The user can add a new group by clicking the ‘+’ icon in the top menu. This action takes the user to the device selection screen.

The *device selection screen* (Figure 2b) shows labeled icons for all available sensor sources and deemphasizes unavailable ones. On selecting a device, the user is taken to a secondary screen for choosing the placement of the device on the body (Figure 2c). Annotating a gesture with the on-body location disambiguates scenarios in which a device can be mounted in multiple locations. A location is fixed by tapping on it on the silhouette.

The designer finally arrives at the *gesture creation/testing screen* (Figure 2d). This view is divided into three sections: a visualization of sensor data on top; a list of gesture classes in the middle; and a test zone at the bottom. If the designer opened an existing group, the gesture classes in the group populate the list. For a new group, a default gesture class is provided. The designer can add new classes by tapping on the ‘+’ icon at the top. Each gesture class is represented with a user-defined name, a count of recorded samples, the ‘+’ icon to add new samples, and a ‘>’ icon for reviewing existing samples.

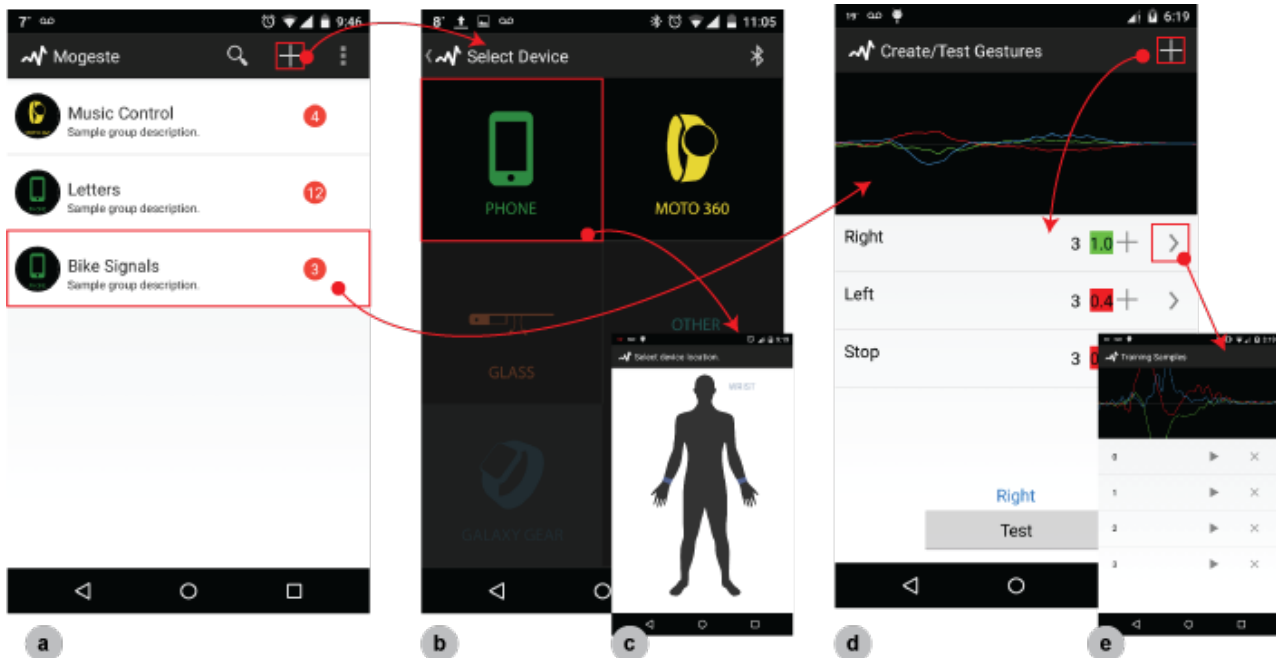


Figure 2. Interface is split in three main screens: a) home screen doubling as gesture group list; b) device selection screen with c) on-body placement selection; d) gesture creation and test screen; and e) sample review screen.

When recording is initiated for a sample, a line chart at the top provides live feedback using a line for each sensor stream. Additionally, a progress bar in the list provides feedback on the time elapsed. Each gesture sample is two seconds long. After at least one sample for each class has been recorded, the designer can test the gesture recognition. She does this by tapping the ‘Test’ button and performing a gesture. The recognition system classifies the gesture and provides both auditory and textual output. Since creation and testing occurs on the same screen, the designer can quickly switch between the two modes. This allows for rapid iteration on a large number of gestures.

Gesture Analysis

In Mogeste, analysis is closely integrated with the design and testing phase. Mogeste presents the designer with an inter-class gesture “goodness” score [5], whenever a new motion is performed. A class’ goodness is based on the F-score (a combined metric representing precision and recall) using 10-fold cross-validation of the class compared to every other class in a gesture group. The goodness values range from 0–100%, and color-coding is used for quick visual feedback (red < 70% < green). Higher values denote uniqueness of a class.

A designer might also want to review the collected samples for a variety of reasons, such as to remove the most recently collected sample or visually inspect patterns of sensor data. The review screen can be accessed by tapping the ‘>’ icon in the gesture class list. This action takes the designer to the *sample review screen* (Figure 2e). The list on this screen is populated by the samples in the class. Selecting a sample shows a line chart visualization playback of the sensor data. The designer can also remove a mistaken sample by pressing the ‘X’ icon in the list item.

The designer is able to switch rapidly between gesture creation and testing on a single screen to iterate on designs. User-defined labels and the ability to review gesture samples provide a means for recollection. System state is made visible through a graphical summary of the data model and a visual progress wheel while recording samples.

Implementation




Mogeste is written using the Android SDK v5.0. We tested it on a Nexus 4 smartphone that connects to a Moto360 smart watch through the Android Wear API, and a Google Glass (2nd edition) eyewear through the Glass Development Kit (GDK). Each of the three devices has a 3-axis accelerometer and a 3-axis gyroscope, operating at the highest sampling rates of 200 Hz, 100 Hz, and 30 Hz, respectively. Inter-device communication is via Bluetooth and connections are managed automatically. Data is stored in the JSON format on a database on the phone.



Mogeste uses a Sequential Minimal Optimization (SMO) implementation of Support Vector Machines (SVM) for multi-class classification and the LibSVM [8] implementation for unary classification from Weka [17] for

Android (<https://github.com/rjmarsan/Weka-for-Android>). SVM requires minimal parameter tuning while still performing well with limited training samples and a high number of features. Moreover, analysis performed by both Bulling et al. [6] and Wu et al. [43] positions SVM ahead of commonly used alternatives found in the literature, such as Dynamic Time Warping (DTW) [15] and Hidden Markov Models (HMM) [34]. We extract the following features per sensor channel: root mean square (RMS), standard deviation (SD), mean, and normalized sensor energy.

Examples

We describe three naturalistic settings below in which Mogeste can be used. In all cases, the gesture group and the class labels can be entered beforehand to save time.

When designing hand-signaling gestures (e.g., left , right , brakes ) for biking, ideally a designer would like to collect live samples. A smart watch paired with a phone mounted on the handlebar (see Figure 1b) could be used as the sensor. In this way, factors such as traffic, and motion artifacts can be accounted for in the examples.

Similarly, consider the case of hands-free operation of a music player application on a smart phone during a run. An eyewear paired with a smartphone strapped to the upper arm could be used for data collection for head gestures such as nodding , side tilt , etc. A designer can now approximate end-use conditions such as sweat, fatigue, and social appropriateness.

Lastly, a designer might want to explore subtle gestures for sending hidden notifications such as notify a friend of your location out during an awkward social encounter. Either a custom ring sensor (similar to Nyenya [3]) that detects finger movement patterns or a behind-ear sensing device [38] that senses distinct jaw motions can be used. The smartphone will be held in a hand. Like before, now subtleness of gestures, consequences of false triggers and mishits, social appropriateness, and ease of use can be easily factored in.

CONCLUSION AND FUTURE WORK

Mogeste’s mobility, simple UI, and novel features enable designers to create diverse gestures in complex naturalistic settings through simple demonstration, fulfilling the criteria set forth by Hudson and Mankoff [23] for effective tools. Furthermore, through Olsen’s situation-task-user (STU) framework [32] we characterize Mogeste as extending the task of motion gesture design to mobile situations (e.g., sitting, running) for interaction designers who would otherwise be hesitant to participate due to their lack of expertise in programming and pattern recognition. Finally, Mogeste refocuses the designers’ time and effort on gesture design by leveraging programming by demonstration, and promoting an iterative workflow.

Future work includes extensive user testing, connecting gestures to application commands, and development of other visualizations. However, Mogeste should spur further interest in the area of motion gesture design.

REFERENCES

1. Abowd, G.D. 2012. What next, ubicomp?: celebrating an intellectual disappearing act. *Proc. Ubicomp*, 31–40.
2. Amma, C., Georgi, M. and Schultz, T. 2014. Airwriting: A Wearable Handwriting Recognition System. *Personal Ubiquitous Comput.* 18, 1: 191–203.
3. Ashbrook, D., Baudisch, P. and White, S. 2011. NENYA: Subtle and Eyes-free Mobile Input with a Magnetically-tracked Finger Ring. *Proc. CHI*, ACM, 2043–2046.
4. Ashbrook, D.L. 2010. Enabling mobile microinteractions.
5. Ashbrook, D. and Starner, T. 2010. MAGIC: A Motion Gesture Design Tool. *Proc. CHI*, 2159–2168.
6. Bulling, A., Blanke, U. and Schiele, B. 2014. A tutorial on human activity recognition using body-worn inertial sensors. *CSUR* 46, 3: 33.
7. Buxton, W. 1986. Chunking and phrasing and the design of human-computer dialogues. *IFIP Congress*, 475–480.
8. Chang, C.-C. and Lin, C.-J. 2011. LIBSVM: A Library for Support Vector Machines. *TIST* 2, 3: 27:1–27:27.
9. Chen, X., “Anthony,” Schwarz, J., Harrison, C., Mankoff, J. and Hudson, S.E. 2014. Air+Touch: Interweaving Touch & In-air Gestures. *Proc. UIST*, 519–525.
10. Cohn, G. et al. 2012. An ultra-low-power human body motion sensor using static electric field sensing. *Proc. Ubicomp*, 99–102.
11. Cohn, G., Morris, D., Patel, S. and Tan, D. 2012. Humantenna: using the body as an antenna for real-time whole-body interaction. *Proc. CHI*, 1901–1910.
12. Cypher, A. and Halbert, D.C. 1993. *Watch what I do: programming by demonstration*. MIT press.
13. Dey, A.K., Hamid, R., Beckmann, C., Li, I. and Hsu, D. 2004. a CAPpella: programming by demonstration of context-aware applications. *Proc. CHI*, 33–40.
14. Fails, J. and Olsen, D. 2003. A design tool for camera-based interaction. *Proc. CHI*, 449–456.
15. Fu, A.W.-C., Keogh, E., Lau, L.Y., Ratanamahatana, C.A. and Wong, R.C.-W. 2008. Scaling and time warping in time series querying. *VLDB Journal* 17, 4: 899–921.
16. Gupta, S., Morris, D., Patel, S. and Tan, D. 2012. SoundWave: Using the Doppler Effect to Sense Gestures. *Proc. CHI*, 1911–1914.
17. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. and Witten, I.H. 2009. The WEKA data mining software: an update. *SIGKDD* 11, 1: 10–18.
18. Hartmann, B., Abdulla, L., Mittal, M. and Klemmer, S.R. 2007. Authoring Sensor-based Interactions by Demonstration with Direct Manipulation and Pattern Recognition. *Proc. CHI*, 145–154.
19. Hinckley, K. et al. 2010. Pen+ touch= new tools. *Proc. UIST*, 27–36.
20. Hinckley, K. et al. 2014. Sensing Techniques for Tablet+Stylus Interaction. *Proc. UIST*, 605–614.
21. Hinckley, K., Pierce, J., Sinclair, M. and Horvitz, E. 2000. Sensing techniques for mobile interaction. *Proc. UIST*, 91–100.
22. Hinckley, K. and Song, H. 2011. Sensor synaesthesia: touch in motion, and motion in touch. *Proc. CHI*, 801–810.
23. Hudson, S.E. and Mankoff, J. 2014. Concepts, Values, and Methods for Technical Human–Computer Interaction Research. In *Ways of Knowing in HCI*. 69–93.
24. Hutchins, E., Hollan, J. and Norman, D. 1985. Direct Manipulation Interfaces. *Human-Computer Interaction* 1, 4: 311–338.
25. Klemmer, S.R., Sinha, A.K., Chen, J., Landay, J.A., Aboobaker, N. and Wang, A. 2000. Suede: a Wizard of Oz prototyping tool for speech user interfaces. *Proc. UIST*, 1–10.
26. Kwapisz, J.R., Weiss, G.M. and Moore, S.A. 2011. Activity Recognition Using Cell Phone Accelerometers. *SIGKDD* 12, 2: 74–82.
27. Long, A.C., Landay, J.A. and Rowe, L.A. 2001. Those look similar! Issues in automating gesture design advice. *Proc. PUI*, 1–5.
28. Lü, H. and Li, Y. 2012. Gesture Coder: A Tool for Programming Multi-touch Gestures by Demonstration. *Proc. CHI*, ACM, 2875–2884.
29. Lü, H. and Li, Y. 2013. Gesture Studio: Authoring Multi-touch Interactions Through Demonstration and Declaration. *Proc. CHI*, ACM, 257–266.
30. Mistry, P. and Maes, P. 2009. SixthSense: a wearable gestural interface. *SIGGRAPH Sketches*, 11.
31. Negulescu, M., Ruiz, J., Li, Y. and Lank, E. 2012. Tap, Swipe, or Move: Attentional Demands for Distracted Smartphone Input. *Proc. AVI*, 173–180.
32. Olsen Jr, D.R. 2007. Evaluating user interface systems research. *Proc. UIST*, 251–258.
33. Patel, K., Fogarty, J., Landay, J.A. and Harrison, B. 2008. Investigating Statistical Machine Learning As a Tool for Software Development. *Proc. CHI*, ACM, 667–676.
34. Rabiner, L. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* 77, 2: 257–286.
35. Rekimoto, J. 2001. Gestur wrist and gestur pad: Unobtrusive wearable interaction devices. *Proc. ISWC*, 21–27.
36. Rubine, D. 1991. *Specifying gestures by example*.
37. Ruiz, J., Li, Y. and Lank, E. 2011. User-defined Motion Gestures for Mobile Interaction. *Proc. CHI*, 197–206.
38. Sahni, H. et al. 2014. The Tongue and Ear Interface: A Wearable System for Silent Speech Recognition. *Proc. ISWC*, ACM, 47–54.

33. Schlömer, T., Poppinga, B., Henze, N., and Boll, S. Gesture Recognition with a Wii Controller. *Proc. TEI* 2008, 11–14.
40. Song, J. et al. 2014. In-air Gestures Around Unmodified Mobile Devices. *Proc. UIST*, 319–329.
41. Starner, T., Auxier, J., Ashbrook, D. and Gandy, M. 2000. The gesture pendant: A self-illuminating, wearable, infrared computer vision system for home automation control and medical monitoring. *Proc. ISWC*, 87–94.
42. Wobbrock, J.O., Morris, M.R. and Wilson, A.D. 2009. User-defined gestures for surface computing. *Proc. CHI*, 1083–1092.
43. Wu, J., Pan, G., Zhang, D., Qi, G. and Li, S. 2009. Gesture Recognition with a 3-D Accelerometer. In *UIC*, D. Zhang, M. Portmann, A.-H. Tan, and J. Indulska, eds. 25–38.
44. Zhang, C., Guo, A., Zhang, D., Southern, C., Arriaga, R. and Abowd, G. 2015. BeyondTouch: Extending the Input Language with Built-in Sensors on Commodity Smartphones. *Proc. IUI*, 67–77.