GestureWiz: A Human-Powered Gesture Design Environment for User Interface Prototypes

Maximilian Speicher, Michael Nebeling
University of Michigan School of Information
{ mspeiche, nebeling }@umich.edu

Figure 1. GestureWiz provides a rapid prototyping environment for gesture-based interfaces via a record–recognize–run pattern: record 2D/3D gestures using a video-based record–replay tool to form mouse, multi-touch, multi-device, and full-body gesture sets (left), use Wizard of Oz optionally powered by crowds to recognize gestures from a given set (middle), and run the resulting human-powered recognizer in user interface prototypes (right).

ABSTRACT

Designers and researchers often rely on simple gesture recognizers like Wobbrock et al.'s $1 for rapid user interface prototypes. However, most existing recognizers are limited to a particular input modality and/or pre-trained set of gestures, and cannot be easily combined with other recognizers. In particular, creating prototypes that employ advanced touch and mid-air gestures still requires significant technical experience and programming skill. Inspired by $1's easy, cheap, and flexible design, we present the GestureWiz prototyping environment that provides designers with an integrated solution for gesture definition, conflict checking, and real-time recognition by employing human recognizers in a Wizard of Oz manner. We present a series of experiments with designers and crowds to show that GestureWiz can perform with reasonable accuracy and latency. We demonstrate advantages of GestureWiz when recreating gesture-based interfaces from the literature and conducting a study with 12 interaction designers that prototyped a multimodal interface with support for a wide range of novel gestures in about 45 minutes.

Author Keywords

Gesture-based interfaces; rapid prototyping; Wizard of Oz; crowdsourcing.

ACM Classification Keywords

H.5.2. Information interfaces and presentation: Input devices and strategies; Interaction styles.

CHI 2018, April 21–26, 2018, Montréal, QC, Canada.

ACM ISBN 978-1-4503-5620-6/18/04 ...$15.00.
http://dx.doi.org/10.1145/3173574.3173681

INTRODUCTION

With the proliferation of many new types of devices and input sensors, such as AR-capable phones and HoloLens, designers face an increased need to support novel forms of touch and gesture-based interaction. While there is an increasing range of gesture recognition tools for developers [17, 29, 30], support for designers is still limited [33]. First, recognizers are constrained by what is feasible with current technology and not necessarily determined by what is desired by users [32]. Second, during design, gestures typically vary both in fidelity and modality, and gesture sets can quickly grow in complexity and raise potential for ambiguity. In many cases, this poses very different technical requirements. For example, if the designer wants to support finger instead of full-body gestures, this also requires different sensing hardware and different models and algorithms at the system level [25].

There is an important research trend to obtain gesture sets from users, rather than designers, through elicitation studies [32]. In these studies, system designers prompt the end-users to demonstrate gestures they would like to use to execute a given system command, and user agreement determines the most popular gestures that make up the set [22, 32]. Apart from less constrained and less biased designs [21, 32], researchers have argued for other advantages to this participatory design approach in terms of memorability [23] and personalization [26]. While there are many advantages for the design, there are also new challenges for the implementation.

Most studies end with elicitation and do not consider implementation in actual recognizers. This raises two issues. First, significant development effort can be required to mitigate ambiguity and conflicts in user-defined gesture sets. For example, a gesture set elicited for a living-room TV web-browsing scenario [20] required significant refinement and, in some cases, substitution with alternative gestures to provide support in an actual Kinect-based system [25]. Second, multiple

input sensors may be required for the implementation. For example, a recent study with 20 participants produced a set of the 44 most frequently suggested interactions for 40 common interaction tasks in AR interfaces [28]. Providing an implementation for this gesture set would require both finger and hand gesture recognition as well as virtual and physical object tracking. User interface prototypes using this gesture set cannot be done without extensive engineering, and may not even be feasible with existing toolkits and gesture recognizers.

We present *GestureWiz*, a gesture-based interface prototyping environment that provides *a)* means to rapidly record *multimodal gestures*, e.g., multitouch, Kinect, Leap Motion, and mid-air 3D gestures; *b)* a human-powered gesture recognizer that works without system implementation and instead uses a *Wizard of Oz* (WOZ) or *crowdsourcing* approach; and *c)* an integrated crowd-powered gesture *ambiguity checker*. A particular strength of our system is the flexibility to use different types of human recognizers. We support using a local Wizard or online crowds if the recognition workload would be too large for a single human. For example, AR research has previously experimented with WOZ approaches [6, 7, 19], but found the six degrees of freedom too difficult to handle.

In contrast to much of the existing crowdsourcing work, our goal is *not* to establish that crowds can also do gesture recognition. Rather, we focus on designers and want to study in which ways they can be supported by other humans in gesture design and evaluation tasks through a combination of WOZ and crowdsourcing. Inspired by earlier work [16], we see a particular promise in making use of humans in experimental gesture interfaces. Until recently, there was no recognition system that could support multiple modalities using different sensor data, e.g., from Kinect or Leap Motion. Only a recent tool called Jackknife [30] comes with a suite of recognizers that can work with few samples and many modalities. Being targeted at developers, it requires that input is represented as a sequence of discrete points in time. For designers who want to experiment with multimodal gestures, there is no simple yet flexible tool. Our goal for GestureWiz is to be such a tool.

We start by presenting crowdsourcing experiments that informed the design of GestureWiz. We then describe our system, its gesture design environment, and a client-side library that enables designers to embed a human-powered recognizer into any gesture-based interface with minimal configuration effort. As part of the evaluation, we demonstrate the use of GestureWiz by re-implementing three challenging gesture-based interfaces from the literature. Finally, we present a study with 12 interaction designers who were able to design novel and conflict-free gesture sets for a given example slideshow application with GestureWiz in about 45 minutes.

RELATED WORK

Our work on GestureWiz builds on research in gesture-based interface prototyping tools and crowd-powered recognizers.

Gesture-Based Interface Prototyping Tools

The research community has experimented with many different tools to support gesture prototyping. For example, Gesture Coder [17] and Gesture Studio [18] are two tools that lower the threshold for developing multitouch gesture recognition code by building on programming by demonstration techniques. DejaVu [9] is an IDE extension that enables programmers to easily record, review, and reprocess temporal data to iteratively improve the processing of Kinect camera input. MAGIC [2] is a motion gesture design tool that provides facilities for experimenting with motion gestures. A key feature of MAGIC is retrospection, allowing designers to review previous actions by visualizing recorded gestures and making a video recording available. GestureAnalyzer [8] provides support for interactive hierarchical clustering of gesture data based on multiple-pose visualisations. It was specifically designed to support researchers in performing elicitation studies, which is only one of the applications supported by GestureWiz. Our goal with GestureWiz was to provide a general input-agnostic gesture prototyping environment that avoids the need for writing gesture recognition code and training recognizers by relying on crowdsourcing instead.

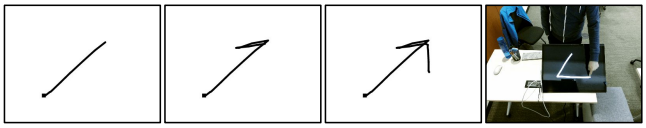

Figure 2. Output of our gesture recording software: *33% Complete*, *67% Complete*, *Complete* (f.l.t.r., animated GIF omitted); plus an example from the additional *Video* template set.

Crowd-Powered Recognizers

Crowd recognition as one of GestureWiz's features is central to much of the crowdsourcing work reported in the literature. Prominent examples include VizWiz [4] and Adrenaline [3], which use camera-based recognition approaches and explored pre-recruiting models such as retainer to keep workers on hold and reduce response times. More specifically related to GestureWiz, researchers have experimented with using crowdsourcing to produce gesture sets, extract features, and train recognizers. For example, Gesture Marks [27] and CrowdLearner [1] used crowds to develop and train gesture sets for mobile applications. Gesture recognition similar to GestureWiz was previously explored in systems like Glance [13], where it was however limited to offline video. GestureWiz adopts many of the principles of these systems, but explores live recognition with crowds by exploring new ways of pushing tasks to workers as new gestures are performed.

The closest to GestureWiz are Apparition [14] and Zensors [12] that coordinate workers in interface design tasks or turn them into camera-based sensors where current technology fails. Yet, there are significant differences: *a)* Zensors is aimed at questions on a higher level of abstraction than required for gesture recognition; *b)* also, its "near real-time" [12] capabilities of snapshots "every one or two seconds" [12] is not sufficient since gestures require a much higher sampling rate; and *c)* different from these two systems, GestureWiz recognition supports multimodal gestures, live streaming, auto-looping, and conflict checking.

DESIGNING A CROWD-POWERED RECOGNIZER

Our goal is to enable designers to rapidly define and test gesture sets using different input modalities, and to have them

recognized by crowd workers or a WOZ rather than an algorithm. As a necessary first step to achieve this—and to get a sense of the requirements posed by such a prototyping environment—we conducted 6 initial experiments to inform the design of our interfaces and get a feeling for the gesture recognition capabilities of crowds under different conditions. During the experiments, we varied three variables: the gestures to be recognized (*test set / test gestures*), the gestures to select the correct match from (*template set / template gestures*), and the UI crowd workers were presented with.

In all experiments, we used the $1 gesture set defined by [33] since its 16 gestures are well-established, well-studied, and therefore provide a good baseline. To create gesture sets, we built on a touch screen and a recording software we specifically implemented for that purpose. For each gesture, the software saved partial gestures (Fig. 2, left) as well as the complete gesture in terms of a static PNG image and an animated GIF image. As for the template sets, we carefully reproduced the $1 single-stroke gestures using that software. To also simulate 3D gestures, we recorded an additional set of videos of a designer performing the gestures on a touch screen (Fig. 2, right). This left us with a total of three static template sets (referred to in the following as *33% Complete, 67% Complete,* and *Complete*) and two animated template sets (referred to in the following as *Animated* and *Video*). As for the test sets, we asked a student to reproduce the $1 single-stroke gestures using our software and recorded on the first try, as if they were using the gestures in an actual application.

1st Iteration: Original UI

In the first iteration, we varied both the test set and the template set (Tab. 1) in order to investigate crowd workers' performance with regard to incomplete gestures and whether animated gestures have an effect on recognition time (as opposed to static ones). The crowd workers were presented with a simple comparison UI (referred to as the *Original UI* in the following) displaying one by one the queue of gestures from the test set on the left, and the complete template set on the right, from which the worker had to select the correct matches by clicking the corresponding gesture (Fig. 3).

After being recruited on MTurk, 200 crowd workers recognized a total of 1873 gestures in the four experiments.[1] Our results (Tab. 1) reveal two particular findings. First, crowd workers are reasonably good at recognizing incomplete gestures, with an accuracy of more than 75% for the test set that is only one third complete and the animated templates. One potential reason for workers being less accurate with the static templates in this case might be that animated gestures also show the incomplete states at some point, while the static ones do not. Second, based on our Original UI, crowd workers were significantly slower at recognizing complete and animated gestures when being presented with an animated template set.[2] We hypothesize that this is due to the large amount of gestures on the right-hand side of the UI, which causes much more noise in the animated than in the static case; and particularly when being confronted with 3D video instead of single-stroke gestures. While differences in accuracy are as well significant when comparing the static to the animated templates, results are still reasonably good, with a minimum of 88% accuracy (static test set / video templates).

Based on these results, we decided for a second iteration of experiments, altering the UI as an additional variable.

2nd Iteration: 1 vs. 1 UI

In a second round of experiments, we investigated accuracy and recognition times of the crowd when being confronted with different user interfaces and the *Complete* and *Animated* test sets as well as the *Animated* and *Video* template sets. For this, we created a second UI, named the *1 vs. 1 UI*, which splits a template set of N gestures into N pair-wise comparisons of the gesture to be recognized and the template gestures. In this way, N crowd workers compare to one template gesture each rather than having one crowd worker compare to all N templates. A 1-on-1 comparison is the most basic, atomic unit and presents the smallest possible cognitive load to the worker. Hence, with this set-up, we intended to bring down recognition times for animated templates while at least retaining the accuracy from the previous experiments.

A total of 1846 crowd workers recruited on MTurk recognized the same amount of gestures in experiments 5 & 6. Our results (Tab. 2) show that based on the new UI, the crowd was able to recognize gestures significantly more accurate in three out of four cases, with accuracy being always above 90%. Moreover, they were also significantly faster in all combinations of test and template sets. This confirms our hypothesis that splitting animated template sets clearly reduces noise for the worker—particularly in the case of 3D video gestures—and therefore makes for a better performance in terms of latency while at least retaining accuracy.

Implications

The above results yield three implications that inform the design of the GestureWiz gesture design environment.

First, it seems feasible to use crowds for gesture recognition in a prototyping scenario, from both a latency and in particular an accuracy standpoint. While GestureWiz can be slower and less accurate than automatic techniques (e.g., $-family with stroke gestures), such comparisons are of limited usefulness. Our approach supports gestures for which no automatic techniques exist yet and is moreover a design environment, not just a recognizer.

Second, we have learned that crowds are good at guessing and resolving ambiguities in terms of incomplete gestures. Therefore, in our prototyping environment, we will rely on live streaming gestures as soon as they begin. In this way, crowd workers can potentially recognize gestures even before their articulation is completed.

Third, since workers had difficulties with animated template gestures in our Original UI, we will include both interfaces and provide designers with the option to choose the 1 vs. 1 UI for recognition. We will also show hints that recommend

[1] In all results, outliers were excluded using *Tukey's test for outliers.*

[2] Latency was tested for significance based on *Mann–Whitney U tests*, while accuracy was tested using χ^2 *tests* ($\alpha = .05$).

#	template set / test set ↓	Complete accuracy (σ)	Complete time [s] (σ)	Animated accuracy (σ)	Animated time [s] (σ)	Video accuracy (σ)	Video time [s] (σ)
1	33% Complete	.55 (.50)	6.01 (3.17)	.76 (.43)	7.43 (4.02)	—	—
2	67% Complete	.80 (.40)	6.06 (3.34)	.89 (.31)	5.09 (2.78)	—	—
3	Complete	.98 (.16) $\star\circ$	3.22 (1.08) $\bullet\diamond$	.91 (.29) $\star$	4.14 (1.95) $\bullet$	.88 (.33) $\circ$	8.31 (4.86) $\diamond$
4	Animated	.92 (.27)	5.03 (1.61) $*\triangleleft$	.93 (.26)	5.89 (2.41) $*$	.92 (.27)	10.95 (5.95) $\triangleleft$

Table 1. Accuracy and latency in experiments 1–4, based on our Original UI ($\star$ $p < .01$, $\circ$ $\bullet$ $\diamond$ $*$ $\triangleleft$ $p < .001$).

#	template set / test set ↓	Animated accuracy (σ)	Animated time [s] (σ)	Video accuracy (σ)	Video time [s] (σ)
	Original UI				
3	Complete	.91 (.29) $\star$	4.14 (1.95) $\diamond$	.88 (.33) $\circ$	8.31 (4.86) $\triangleleft$
4	Animated	.93 (.26)	5.89 (2.41) $*$	.92 (.27) $\bullet$	10.95 (5.95) $\triangleright$
	1 vs. 1 UI				
5	Complete	.98 (.13) $\star$	2.22 (0.92) $\diamond$	.98 (.13) $\circ$	3.38 (1.05) $\triangleleft$
6	Animated	.91 (.29)	2.97 (1.37) $*$	.97 (.16) $\bullet$	4.01 (1.69) $\triangleright$

Table 2. Accuracy and latency in experiments 3–6, based on our Original UI and the 1 vs. 1 UI ($\bullet$ $p < .01$, $\star$ $\circ$ $\diamond$ $*$ $\triangleleft$ $\triangleright$ $p < .001$).

the use of the latter when the designed gesture set is not solely comprised of static gestures.

THE GESTUREWIZ PROTOTYPING ENVIRONMENT

The GestureWiz prototyping environment consists of three main components: a **Requester UI** for recording gestures and conflict checking; the **Worker UIs** (Original and 1 vs. 1) for recognizing recorded gestures using crowds or a WOZ; and the **GestureWiz library** to be used in applications that shall be enhanced with gesture recognition capabilities.

The GestureWiz workflow for a designer to define and test a gesture set and their own application prototype is as follows:

(a) First, they can use the Requester UI in *record* mode and capture a set of template gestures using one or more types of input, then save them.

(b) In the next step—potentially consisting of several iterations—the gesture set can be tested using the built-in *conflict checker* to resolve ambiguities and adjust the design of the gestures.

(c) Subsequently, after extending the (optional) application prototype to be controlled with the gestures with the GestureWiz library, arbitrary commands within the application can be mapped to any of the defined gestures.

(d) Finally, test gestures can be streamed for recognition by crowd workers or a WOZ who compare them to the previously saved templates, using any of the two available Worker UIs. Streaming happens either through the Requester UI's *recognition* mode or a custom solution based on the GestureWiz library. As soon as a gesture is recognized, the corresponding command will be executed in the application based on an asynchronous callback.

In the following, we explain our environment in more detail along the lines of this workflow.

(a) Recording Gesture Sets using Multiple Inputs

GestureWiz's Requester UI (Fig. 3) supports recording gestures using a range of different input modalities. Currently, mouse, pen, and multitouch strokes, as well as Kinect body stream, Leap Motion, and video are supported (Fig. 3d). Due to the flexible design of the prototyping environment, this can be easily extended to new inputs. To do so, GestureWiz supports stream capture of anything drawn on an HTML5 canvas element (Fig. 3b), from which it then automatically generates video sequences that function as gesture templates.

Supplying Alternative Gesture Templates

New gesture templates immediately appear in the Requester UI (Fig. 3f), allowing users to preview them in the way crowd workers would see them in recognition mode. Templates the designer is not satisfied with can be easily discarded using the red minus button. This allows users to batch record several templates of the same gesture, compare them visually in the Requester UI or using the conflict checker, and keep the template that best represents the intended gesture. Gesture sets can be stored under a user-supplied name and reloaded, allowing gesture recording in multiple independent sessions and experimentation with different gesture sets.

Dealing with Gesture Representation Problems

Note that it is possible to produce multimodal gesture sets by recording gestures using different inputs as part of the same template set. This can be used to supply templates of different types of gestures, e.g., surface vs. mid-air gestures, if the application is to support multiple input modalities. However, this can also be used to supply templates of the *same* gesture type in different representations, e.g., Kinect body vs. camera video sequences, for mid-air 3D gestures. This supports experimentation if it is not clear which representations are the fastest to be recognized and matched by crowd workers.

Dealing with Gesture Segmentation Problems

Note that in record mode, stream capture for gesture recording is manually controlled by the user via the space bar or start/stop button. This not only allows for segmentation of continuous interaction streams supplied by Kinects and cameras, but also for recording gestures consisting of multiple mouse, pen, or touch strokes. Recognition mode, however, can work on a live stream. Crowds or WOZ simply select a template as soon as they recognize a gesture in the stream.

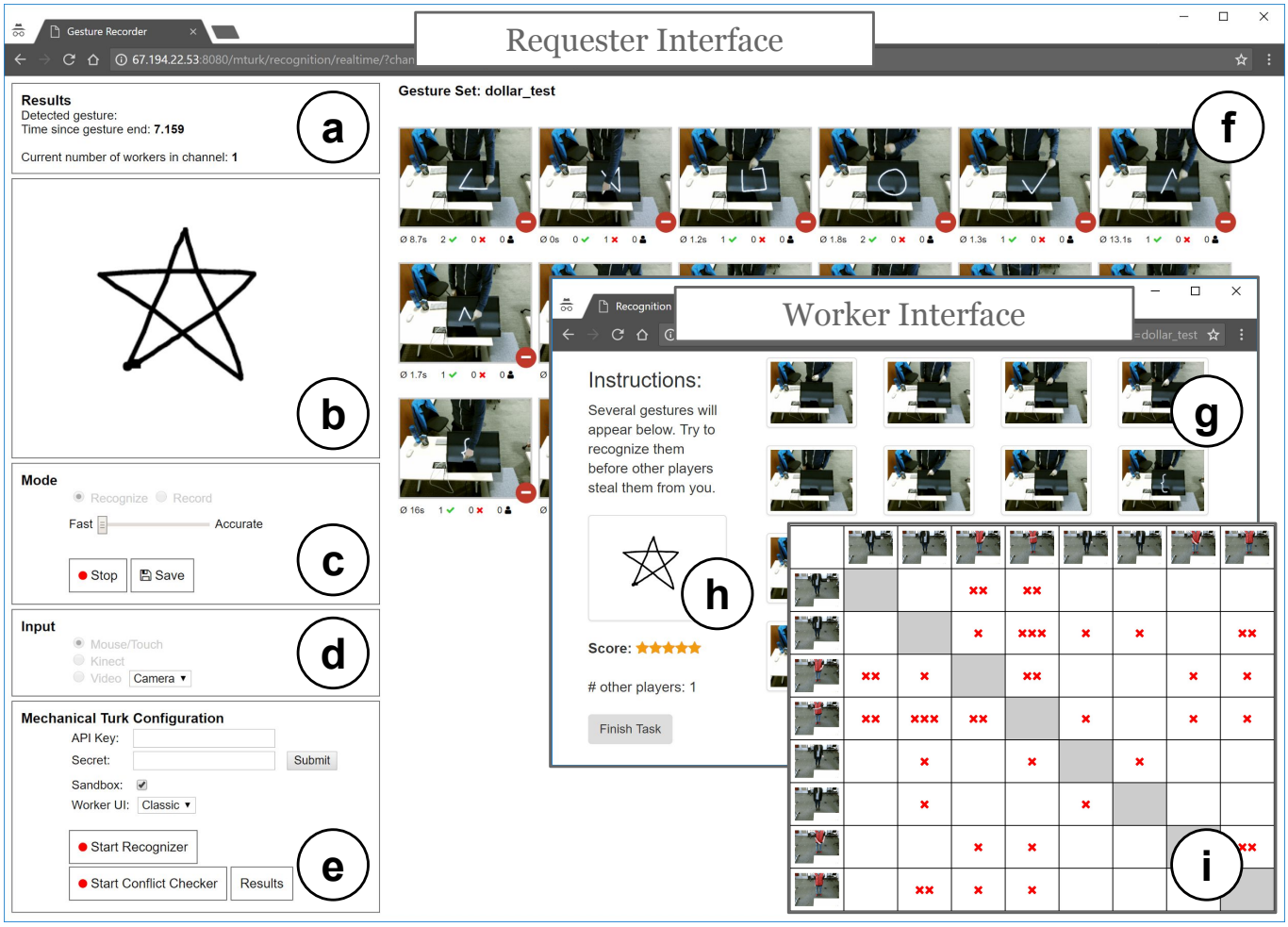

Figure 3. GestureWiz's requester and worker UIs: *(a)* recognized gesture, time taken, and number of workers, *(b)* input canvas (here showing camera video), *(c)* mode switch to record/recognize gestures, *(d)* input modality, *(e)* MTurk configuration, *(f)+(g)* gesture templates, *(h)* requester live video, and *(i)* integrated conflict checker (3 X's maximum).

(b) Resolving Ambiguities Using Conflict Checking

The integrated conflict checker automatically posts pair-wise comparisons of all non-identical gestures in the template set to MTurk, i.e., $N(N-1)$ comparisons for N gestures. Currently, each comparison (A, B) is assigned to three crowd workers. If gesture A is incorrectly recognized as matching gesture B, the pair of gestures is marked as ambiguous (Fig. 3i). This provides valuable early-stage feedback and supports rapid prototyping of less ambiguous gesture sets.

(c) Making Use of the GestureWiz Library

The optional end-user library included in our framework provides two tools. First, a method to set up a custom solution for gesture streaming rather than relying on the requester interface's recognition mode, and second, a means to receive gesture detection events in an application and invoke commands accordingly. Both tools have been designed for minimal configuration effort and can be set up with a minimal amount of code.

(d) Streaming Gestures in Nearly Real-Time

When the user activates the Requester UI's *recognition* mode and records a gesture, it is automatically streamed to the Worker UI (Fig. 3h) rather than being added to the template set. In the Worker UI, crowd workers or a WOZ compare the incoming test gesture to the designer's previously defined template set in case of the Original UI (Fig. 3g), or to just one of the template gestures in case of the 1 vs. 1 UI. When the recording is completed, the Worker UI automatically replaces the stream with a repeating video sequence of the test gesture. In this way, a worker can observe the gesture already while it is being articulated. If a worker enters after the stream was closed, they are instead presented with the repeating video sequence. This combination of live streaming and substitution with a repeating video yields the best chance of a fast worker response. The reason for this is that crowd workers have shown to be good at recognizing incomplete gestures and therefore might correctly identify the gesture even before it is completed by the requester.

To motivate good worker performance, the Worker UIs follow a gamification approach. In the instructions, workers are informed that they are competing against other workers to provoke quick responses. This is realized by removing the test gesture from the Worker UI as soon as an answer has been delivered to the requester and thereby "taking away" the gesture

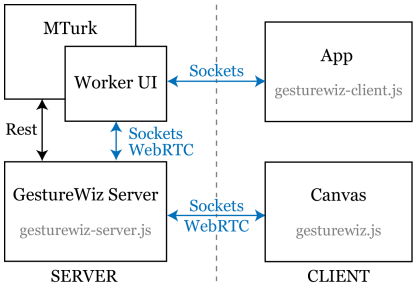

Figure 4. Technical architecture of GestureWiz.

from workers that were too slow. To also ensure accuracy, workers gain points in the form of stars for correct answers, which grants them an according bonus payment in MTurk.

GestureWiz allows for pre-recruiting of crowdworkers already during the gesture design phase. That is, following the approach by [3], they are kept on hold and notified as soon as the designer starts streaming a gesture in *recognition* mode. The currently available number of workers can be monitored in the Requester UI in real-time (Fig. 3a).

Tuning Gesture Recognition Through Different UIs

There are two ways to influence the accuracy and recognition speed of the system. First, the designer can choose between presenting workers with the Original UI or with the 1 vs. 1 UI. The former is slower and slightly less accurate since one worker has to take care of comparing all N gestures of a template set to a test gesture. In the 1 vs. 1 UI, on the other hand, N crowd workers take care of only one comparison each, which makes it quicker and slightly more accurate. The main drawback of the latter is the increased cost due to significantly more tasks that have to be posted to MTurk. Also, it is only feasible in a WOZ set-up if N wizards are available.

Tuning Gesture Recognition Through Mediation Strategies

Second, because of the trade-off between fast and accurate worker responses (independent of the UI), GestureWiz's Requester UI allows designers to tune the gesture recognition settings concerning accuracy and latency. Based on common mediation strategies, we provide three settings: *fast*—the result will be the first response by any worker; *balanced*—the result will be the best of three worker responses; and *accurate*—the result will be the best of five worker responses. If the two latter options cannot determine a single winner, an array of the gestures with the most votes is provided instead.

ARCHITECTURE, IMPLEMENTATION, & USAGE

In the following, we briefly introduce the technical architecture of our prototyping environment, as well as its implementation and how to integrate it into external applications with minimal programming effort.

Architecture

GestureWiz is based on a client–server architecture designed for minimal programming and configuration effort for the end-user. The server handles the communication with MTurk as well as with the workers recruited for recognizing gestures. On the client side, there is the Requester UI for prototyping gesture sets, which is an integral part of the prototyping environment and therefore comes with the server. Also on the client side reside one or more UIs owned by the end-user that make use of GestureWiz's gesture recognition capabilities. Both the Requester UI and the end-user UIs directly communicate with the server but neither with one another nor with MTurk. The gesture recognition itself happens asynchronously, which makes the architecture callback-oriented. It is possible to have the same or different UIs send gestures and handle the answers received from workers or the WOZ.

Implementation & Usage

The GestureWizserver was implemented using Node.js. Based on Socket.IO, it manages the communication between the different parts of the system and provides endpoints for invoking various methods of the MTurk API, which are used by our client-side JavaScript library. The Requester UI was realized based on HTML5 and JavaScript and features a canvas for recording template gestures, which are permanently stored on the server along with corresponding identifiers. After including our *gesturewiz.js* library, the end-user initializes the system in their UI by referencing the canvas element recording the gestures that shall be detected as well as the unique identifier of a previously stored set of template gestures. Additionally, they define a callback to be invoked when a gesture has been recognized.

```
const wiz = new GestureWiz().init(canvas, templateSet)
  .ongesture(function(detectedGesture) { /* ... */ });
```

After initialization, gesture recognition is invoked by calling the `recognize()` method, optionally specifying accuracy as a parameter. Once this has happened, via WebRTC the content of the previously referenced canvas is streamed in real time to the Worker UI that is shown to the crowd workers recruited by the server (or the WOZ). Recruiting can either happen manually by using dedicated methods provided by *gesturewiz.js* or automatically by passing an additional parameter to `init()`. Unless the end-user calls the method for deactivating MTurk, the server automatically ensures that the number of active workers stays constant at a predefined value. If the interface handling the detected gestures is different from that sending the gestures, the end-user invokes `toClient("myUI")` instead of `ongesture()`. The other interface then needs to include and initialize our *gesturewiz-client.js* library:

```
const wizClient = new GestureWizClient("myUI")
  .ongesture(function(detectedGesture) { /* ... */ });
```

GESTUREWIZ APPLICATIONS

In this section, we present three applications we created using GestureWiz: *(1)* a **maps viewer** using touch gestures for zooming, *(2)* a **video player** using Kinect hand gestures for playback and volume control, and *(3)* a **watch+phone email app** using cross-device gestures for notification management. Both the input (multitouch, Kinect, video) and the types of gestures (surface vs. mid-air) varied between applications. With these examples, we demonstrate the feasibility and effectiveness of GestureWiz, as well as the range of applications and gesture modalities that can be prototyped.

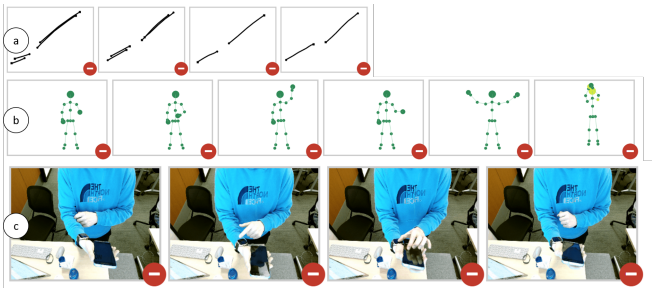

Figure 5. Gestures recorded using GestureWiz for *(a)* a maps app using multitouch zoom in/out gestures (based on [32, 22]); *(b)* a video player using Kinect hand gestures (based on [31, 20, 25]); and *(c)* cross-device gestures from Duet [5].

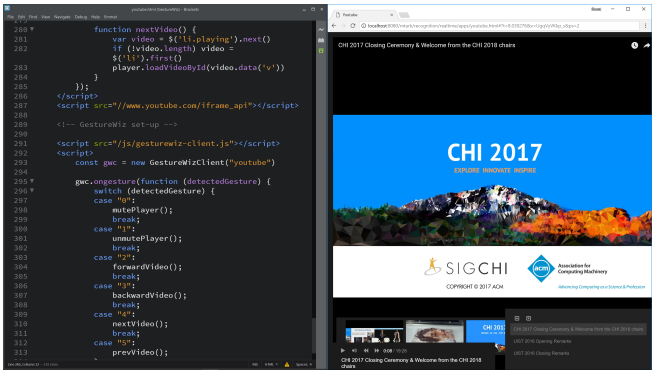

Figure 6. Example integration of the GestureWiz library and mapping of gestures to corresponding functions in our YouTube app.

Multitouch Stroke Gestures with GestureWiz

First, we created a maps viewer using the Google Maps API. Based on guessability studies carried out by Wobbrock et al. [32] and Morris [22], we recreated four surface gestures for zooming the map in and out by recording *multitouch* strokes using GestureWiz. Our set consisted of two pairs of gestures: hand swipe left/right and finger stroke left/right (Fig. 5a). We embedded the GestureWiz library into the application and mapped the corresponding multitouch strokes to zoom in/out functions in the maps viewer.

Kinect Body Gestures with GestureWiz

Second, we created a video player using the YouTube API. Inspired by elicitation studies from Vatavu [31], Morris [20], and Nebeling et al. [25], we recreated six mid-air gestures for volume and playback control by recording *Kinect* body streams using GestureWiz. Our set consisted of three pairs of gestures: flick hands left/right, move hands up/down, and move hands apart/together (Fig. 5b). Again, based on the GestureWiz library, we mapped gestures recognized from Kinect body streams to video seeking and volume control functions in the video player (Fig. 6).

Video Cross-Device Gestures with GestureWiz

Third, we created a cross-device email app adapted from XD-Browser [24] using four cross-device gestures inspired by Duet [5]. Our set consisted of two pairs of swipe and pinch gestures from watch to phone and vice versa (Fig. 5c). These gestures were originally created by researchers to explore joint interactions between watch and phone. We recreated them by recording the interactions in terms of *video* gestures using GestureWiz, then mapped gestures recognized from video streams to visual feedback indicating whether notifications were enabled on each device.

	#	N	accuracy (σ)	t [s] (σ)
$1 unistroke	16	2284	.99 (.18)	3.92 (1.20)
Multitouch	4	152	.81 (.39)	5.62 (1.93)
Kinect	6	348	.91 (.29)	6.59 (4.22)
Video	4	150	.79 (.41)	7.38 (3.89)
total	30	2934	.96 (.20)	4.50 (2.33)

Table 3. Crowd recognition benchmarks for different modalities (# = number of gestures).

Crowd Recognition Experiments

To investigate GestureWiz's robustness w.r.t. different gesture modalities, we report recognition accuracy and latency for each of the three animated gesture sets recreated for the above applications—plus the $1 *Video* gestures from our initial experiments as a baseline. Due to the animated nature of the gesture sets, the experiments are based on comparisons using the 1 vs. 1 UI (Tab. 3). For each application, the authors of this paper carefully reproduced the selected gestures in terms of a template set, performing each gesture very clearly. Additionally, we created corresponding sets of test gestures, which we performed faster and recorded at the first try, in order to simulate real-world gesture input in an application. Then, for each example application with N gestures, we created $N \times N$ 1-on-1 comparisons and recruited 10 crowd workers per comparison. While GestureWiz uses established techniques for recruiting and keeping multiple workers on hold, it does not make use of parallel crowd workflows yet [10, 11]. Using 1-on-1 comparisons with pre-recorded gesture sets, however, we created a benchmark for the performance of GestureWiz as if these techniques were adopted and gesture sets divided into the smallest possible, atomic units. The performance of parallel and instantly available crowd workers performing N comparisons to find a sample gesture in an N-gesture template set should not differ from a single comparison.

In these experiments, we expected a decrease in accuracy and an increase in latency the more complex the gesture set gets. Thus, in Tab. 3, the gesture sets are sorted from least to most complex (least complex at the top), as anticipated before the study. Overall, we investigated 30 gestures that were recognized by a total of 2934 crowd workers (after removing outliers). Results show a minimum accuracy of 79%, which was recorded for the Duet gestures that were recorded as videos and used in the watch+phone email app. The maximum accuracy of 99% was achieved by the $1 unistroke gestures that served as our baseline. The same gesture set also achieved the minimum latency of 3.92 s, while the maximum was again recorded for the Duet gestures (7.38 s). This largely confirms our hypothesis concerning the relationship between complexity, accuracy, and latency, with the exception of the Kinect gestures, which were recognized more accurately than the multitouch set. The most probable reason is that although the Kinect gestures are more noisy and in general seem more

	Mouse/Pen	Touch	Kinect	Camera/Video
mean	4.33	4.33	2.58	2.50
median	5	5	2.5	2

Table 4. Participants expertise with using/designing gestures using different modalities based on a 5-point scale (1=*no knowledge*, 5=*expert*).

complex (due to the larger number of visual features), the multitouch gestures are more ambiguous (Fig. 5), thus potentially leading to more incorrect recognitions. Results for the $1 unistroke gestures moreover confirm our initial experiments since we replicated the set-up with the *Animated* test set and *Video* template set in a 1 vs. 1 setting (Tab. 2, accuracy = 97%, latency = 4.01 s).

In summary, the applications implemented with GestureWiz as well as the experimental results show the robustness and feasibility of our environment in prototyping scenarios that employ a multitude of modalities. We are able to rapidly define established gestures from the literature and seamlessly integrate them into (existing) applications. Overall, it took less than 90 minutes to record the template as well as test sets and integrate the GestureWiz library in all three applications. Based on crowd workers, the prototyped gestures can be recognized with reasonable accuracy and latency even in ambiguous, noisy, and complex cases. Across the different modalities that were investigated, we were able to record an average accuracy of 96% at an average latency of 4.50 s.

STUDY WITH INTERACTION DESIGNERS

To also study the gesture prototyping process based on GestureWiz *in situ* in an exploratory setting, we conducted an additional study with interaction designers, i.e., all participants took at least one course on interaction design in university. Our aim is to show that participants are able to quickly understand and use our prototyping environment for designing and testing a gesture set for a specific use case. Although we as well report recognition performance in the following, the focus of this study lies on the prototyping process. Therefore, we did not specifically optimize the set-up of GestureWiz for maximum accuracy and recognition speed during the study.

Method

The study was divided into four parts—informed consent, gesture definition, conflict resolution, and gesture recognition. It was carried out in teams of two, i.e., we studied six teams of interaction designers, with 12 participants in total (average age of 23.8 years, six male, six female). Each team completed the whole study in about 60 minutes (introduction and informed consent took about 15 minutes in each case). We chose to have participants co-design their gesture sets since firstly, this set-up has proven to be particularly feasible for gesture prototyping [21], and secondly, we juxtaposed WOZ recognition to crowd recognition.

First, we introduced participants to the functionalities of the Requester UI and a simple slideshow application we implemented for the purpose of the study. Subsequently, we asked them to together design pairs of gestures for going to the next/previous slide, using GestureWiz's recording functionality. Using production [21], we encouraged them to try as many suitable gestures as they could think of, including unconventional ones. We limited the modality to mid-air 3D gestures recorded as videos (using Kinect) due to the lack of feasible and effective existing solutions. Such gestures are the most challenging use case for existing approaches and therefore also the most relevant one with respect to GestureWiz. In the second phase, participants were introduced to the integrated conflict checker to test gestures for ambiguities and potentially adjust their gesture sets. After that, the two team members took turns in doing WOZ recognition—one acting as the wizard and one performing live gestures to be recognized. In a final step, both team members performed live gestures that were streamed to and recognized by pre-recruited crowd workers. To ensure comparability of results, we chose the Original UI for both, the WOZ and the crowd set-up. Also, to be able to reliably measure performance, we manually segmented the gestures to be recognized, rather than working with a stream. After that, participants were asked to give feedback based on a post-study questionnaire. The teams produced sets ranging from six to 12 gestures (avg. = 9, median = 9), with a clear majority of swipe, pull, point, and "click" gestures. See Tab. 4 for more demographic information.

Observations

During the whole study process, participants were observed by one investigator while another acted as the study facilitator. In the following, we report the main observations.

(O1) Four out of six teams recorded multiple instances of at least one gesture and then kept only the best one after review, deleting the others.

(O2) In three cases, a designer who performed a live gesture for recognition did a different gesture than intended (e.g., moving their arm left instead of right), stating as the reason that Kinect mirrors the recorded video.

(O3) Although we pre-recruited crowd workers, in three cases the only worker left the Worker UI before the current gesture was recognized. This led to unusually long recognition times.

(O4) Two designers explicitly stated that a particular gesture might have been performed too fast for correct recognition. One did so immediately after recording ("The crowd is going to miss that one.") and one during conflict checking.

(O5) Three teams noted that certain gestures (particularly pointing gestures) might be too small to be recognized correctly. ("Pointing with one finger is probably not clear enough for workers.")

(O6) Five out of six teams agreed with most of the conflicts detected by the conflict checker and subsequently stated they wanted to adjust their gesture set according to the feedback.

(O7) Two teams disagreed with conflicts in which crowd workers got the direction of the gesture wrong (e.g., recognized swipe left as swipe right).

(O8) In relation to O7, for three teams the conflict checker reported conflicts for pairs of gestures that seemed to be mirrored versions of the same gesture (e.g., swipe left/right with different arms).

Statement	mean	median
Felt easy to design new gestures	4.75	5.5
Felt fast to design new gestures	4.67	4.5
Enjoyed designing new gestures	6.17	6
Felt easy to test new gestures	5.50	6
Felt fast to test new gestures	4.83	5
Enjoyed testing new gestures	5.58	6
WOZ accuracy was good enough	5.75	6
WOZ speed was good enough	5.33	5.5
Crowd accuracy was good enough	3.42	3
Crowd speed was good enough	3.83	4
Comparable to existing systems	5.00	5.5

Table 5. Participants' ratings for 11 statements on a 7-point Likert scale.

(O9) Three teams noted that the segmentation of at least one gesture was not as intended and suggested this led to lower recognition accuracy.

(O10) Three teams (in which participants wore significantly different clothing) noted during that crowd workers seemed to be orienting at their clothing for recognition, which led to incorrect results. Yet, for two of the teams, crowd workers still managed to correctly recognize gestures in several cases.

(O11) Three teams decided to not change conflicting gestures that were intended to be different but trigger the same action.

(O12) Three teams suggested that conflicting gestures might be due to unclear starting points.

(O13) Besides straightforward gestures, participants also recorded a number of more unconventional ones, like kicking left/right or single-finger swipes on the opposite forearm.

Feedback

In the post-study questionnaire, participants were asked what they considered as the main benefits (**B**) and limitations (**L**) of GestureWiz. Their answers are summarized in the following.

(B1) Four participants stated that GestureWiz is easy and fast to control (or use, respectively).

(B2) The conflict resolution capabilities were mentioned as a main benefit three times. One designer mentioned that GestureWiz helps with understanding how people perceive gesture similarity.

(B3) Three participants positively noted that the our environment allows (potentially multiple) designers to understand gestures better and get quick feedback.

(B4) One designer mentioned the possibility to test gestures in different ways (WOZ, crowds) as a main benefit. One mentioned the general flexibility of the environment.

(B5) "The system provides good feedback" was mentioned as the main benefit by one participant.

(L1) As a main limitation, two participants referred to privacy issues (regarding video recording and how it affects the choice of gestures).

(L2) Two participants stated that the recognition speed was suboptimal. One mentioned bad accuracy.

	time [s]	σ	accuracy	σ
Wizard of Oz	3.22	2.01	.83	.38
Crowd	5.43	4.30	.46	.50

Table 6. WOZ and crowd recognition performance during the study.

(L3) One designer mentioned that crowd workers pay attention to irrelevant details in the recorded video that are not intended to be part of the gesture.

(L4) Another designer noted that recognition results are affected by crowd workers' motivation.

(L5) The fact that GestureWiz only tests the similarity of gestures, but not whether a gesture is difficult to perform for the user was mentioned as the a limitation by one participant.

(L6) One participant noted that incorrect segmentation is an issue that affects recognition.

We moreover asked the interaction designers to rate different statements on a 7-point Likert scale (Tab. 5). In general, they reported a good experience of designing and testing new gestures using GestureWiz. WOZ accuracy and speed were rated clearly above average while crowds fell behind in this respect. The latency of crowd workers was considered average (median = 4) while their accuracy was rated below average with a median of 3. Finally, participants somewhat agreed with the statement that GestureWiz is comparable to existing systems.

Recognition

Across all teams, we recorded a total of 99 gestures recognized by WOZs and 94 by crowd workers (after removing outliers). Overall, the WOZ set-up provided a better recognition performance concerning both, accuracy and latency (Tab. 6). This seems natural since the wizards had the advantage of trying to recognize gestures they co-designed themselves. While crowd accuracy is below 50% for the study set-up, crowd latency is lower than that of the video gestures in the crowd recognition experiments described earlier, despite being based on the Original UI.

Findings

From the above observations, feedback, and recognition performance, we can derive a number of findings. First, interaction designers generally find our prototyping environment quick and easy to use (B1, Tab. 5) and especially appreciate its conflict resolution capabilities (O6, B2) and the support it provides for designers (B3, B4, B5). Also, participants on average stated that GestureWiz provides an enjoyable experience of designing and testing gestures (Tab. 5). "It is fun! Feedback for recognition is straightforward," was a statement by one designer. Second, whether gestures have to be performed exactly depends on the similarity to other templates and individual differences between human recognizers (L3, L4). In our study, a certain degree of variation was possible since some workers were invariant to orientation (O7, O8) or correctly recognized gestures despite different clothing (O10). Third, interaction designers clearly approve of WOZ recognition capabilities as opposed to crowds (L2, L3, L4), which is supported by the quantitative recognition results. Fourth, however, crowd workers were impaired by

some specific circumstances that were not present during the experiments we reported on earlier. One such factor was that two persons contributed gestures to the same set, which led to the problem of workers treating clothing and other irrelevant details as part of the gesture (O10, L3). One participant described this as "clothes-driven gesture recognition". Moreover, gestures in which two variables were inverted compared to another gesture (e.g., swipe left with right arm vs. swipe right with left arm) and which therefore appeared to be mirrored posed problems to, not only crowd workers (O7, O8), but even some designers (O2). In addition, some designers consciously decided against revising conflicting gestures in case they were intended for the same action anyway (O11). Finally, the gestures recorded during the study were ad-hoc and therefore generally noisy, which as well posed potential problems to the crowd workers (O4, O5, O9, O12, L6).

These findings not only confirm that mid-air 3D gestures are a non-trivial use case, but also provide valuable input for revising and improving GestureWiz. For instance, irrelevant features such as bright colors could be reduced by applying filters to a video loop before showing it to workers. Also, the mirroring issue could be prevented by adding artificial features to the video gestures to make them clearly distinguishable, which is a potential solution proposed by one participant. We will consider this in future versions of GestureWiz.

DISCUSSION & LIMITATIONS

GestureWiz has numerous real-world applications, especially in situations with a lack of required hardware or mixed modalities. One example is a designer who intends to create a set of hand gestures for Kinect and Leap Motion, but does not have the systems available. Another example is an application combining a whiteboard and HoloLens, in which mid-air hand gestures and drawn gestures on the board should be supported. In both cases, GestureWiz enables rapid prototyping of the corresponding gesture sets with just a webcam.

However, our system still has some limitations. First, it is difficult to establish a quantitative baseline for our study. We, among other things, aim at situations for which no reliable algorithmic recognizers exist yet, like arbitrary and noisy mid-air 3D gestures. For instance, [29, 30] are ill-suited for comparison: they only aim at hand gestures or require time-series data rather than raw video. Thus, they would not have worked with many gestures from our study. We want to stress again that such performance comparisons are not central to our goal of providing a gesture design environment.

Second, it would be desirable to automatically learn models from gestures recognized by humans. Zensors [12] trains classifiers based on individual, static image snapshots. Translating this to GestureWiz, however, would mean training classifiers from live video. This requires further research and a different, spatio-temporal approach, i.e., detecting not only where things are, but also how they move between frames in short periods of time ($<$1s for some gestures).

Third, GestureWiz does not yet have dedicated support for dynamic gestures. Yet, our results for partial gestures suggest that early recognition is possible (Tab. 1). For dynamic gestures like drag'n'drop, early recognition could be simulated by recording start and end portions as separate templates. Tracking features like gesture size and orientation would require templates of different size and orientation. A more sophisticated approach would require crowd tracking and remote UI manipulation similar to Legion [15].

CONCLUSION

We have presented the GestureWiz prototyping environment that provides designers with an integrated solution for gesture definition, conflict checking, and recognition in nearly real-time using Wizard of Oz or crowdsourcing set-ups. Informed by initial experiments with crowd workers, we developed optimized interfaces for requesters and workers and suitable strategies for recognition. To prove the feasibility and effectiveness of our approach, we prototyped and tested three applications featuring different input modalities and kinds of gestures, which would typically require multiple gesture recognition libraries and significant programming skills.

An explorative study with 12 interaction designers revealed benefits and limitations of our approach. Participants generally appreciated the possibility to quickly and easily define and test new gestures, as well as the integrated conflict resolution functionality and enjoyed using GestureWiz. All of the interaction designers, in pairs of two, were able to create and test gesture sets from scratch and have them recognized by a WOZ and crowd workers in about 45 minutes. Code and data are available at `https://github.com/mi2lab/gesturewiz`.

Our larger vision behind this work is to enable interactive surfaces and spaces anytime, anywhere. GestureWiz provides an important first step towards this vision. It is now possible to walk into an interactive room, experiment with different kinds of non-trivial gestures and test them on the spot, without having to write a single line of gesture recognition code. In the future, it will be interesting to formalize the gesture definition and recognition approach using crowds to support guessability studies while enabling quick testing and revision using GestureWiz. On the crowdsourcing side, future work needs to investigate how to mitigate the shortcomings related to accuracy and latency that were revealed during the user study. Another direction of work we intend to address is making use of the data provided by WOZ and crowds to automatically train corresponding machine learning models, thus facilitating recognition after the prototyping stage.

Acknowledgments

We thank our study participants, Pallavi Gupta and Jaee Apte for their help with initial versions of the worker interface, and Janet Nebeling for her help with the video and figures.

REFERENCES

1. Amini, S., and Li, Y. CrowdLearner: Rapidly Creating Mobile Recognizers using Crowdsourcing. In *Proc. UIST* (2013).

2. Ashbrook, D., and Starner, T. MAGIC: A Motion Gesture Design Tool. In *Proc. CHI* (2010).

3. Bernstein, M. S., Brandt, J., Miller, R. C., and Karger, D. R. Crowds in Two Seconds: Enabling Realtime Crowd-Powered Interfaces. In *Proc. UIST* (2011).
4. Bigham, J. P., Jayant, C., Ji, H., Little, G., Miller, A., Miller, R. C., Miller, R., Tatarowicz, A., White, B., White, S., and Yeh, T. VizWiz: Nearly Real-time Answers to Visual Questions. In *Proc. UIST* (2010).
5. Chen, X. A., Grossman, T., Wigdor, D. J., and Fitzmaurice, G. W. Duet: Exploring Joint Interactions on a Smart Phone and a Smart Watch. In *Proc. CHI* (2014).
6. Dow, S., Lee, J., Oezbek, C., MacIntyre, B., Bolter, J. D., and Gandy, M. Wizard of oz interfaces for mixed reality applications. In *Proc. CHI Extended Abstracts* (2005), 1339–1342.
7. Dow, S., MacIntyre, B., Lee, J., Oezbek, C., Bolter, J. D., and Gandy, M. Wizard of oz support throughout an iterative design process. *IEEE Pervasive Computing 4*, 4 (2005), 18–26.
8. Jang, S., Elmqvist, N., and Ramani, K. GestureAnalyzer: Visual Analytics for Pattern Analysis of Mid-Air Hand Gestures. In *Proc. SUI* (2014).
9. Kato, J., McDirmid, S., and Cao, X. DejaVu: Integrated Support for Developing Interactive Camera-Based Programs. In *Proc. UIST* (2012).
10. Kittur, A., Smus, B., Khamkar, S., and Kraut, R. E. Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, ACM (2011), 43–52.
11. Kulkarni, A., Can, M., and Hartmann, B. Collaboratively crowdsourcing workflows with turkomatic. In *Proc. CSCW* (2012).
12. Laput, G., Lasecki, W. S., Wiese, J., Xiao, R., Bigham, J. P., and Harrison, C. Zensors: Adaptive, rapidly deployable, human-intelligent sensor feeds. In *Proc. CHI* (2015).
13. Lasecki, W. S., Gordon, M., Koutra, D., Jung, M. F., Dow, S. P., and Bigham, J. P. Glance: Rapidly Coding Behavioral Video with the Crowd. In *Proc. UIST* (2014).
14. Lasecki, W. S., Kim, J., Rafter, N., Sen, O., Bigham, J. P., and Bernstein, M. S. Apparition: Crowdsourced User Interfaces that Come to Life as You Sketch Them. In *Proc. CHI* (2015).
15. Lasecki, W. S., Murray, K. I., White, S., Miller, R. C., and Bigham, J. P. Real-time crowd control of existing interfaces. In *Proc. UIST* (2011), 23–32.
16. Lee, S.-S., Chae, J., Kim, H., Lim, Y.-K., and Lee, K.-P. Towards more Natural Digital Content Manipulation via User Freehand Gestural Interaction in a Living Room. In *Proc. UbiComp* (2013).
17. Lü, H., and Li, Y. Gesture Coder: A Tool for Programming Multi-touch Gestures By Demonstration. In *Proc. CHI* (2012).
18. Lü, H., and Li, Y. Gesture Studio: Authoring Multi-touch Interactions through Demonstration and Declaration. In *Proc. CHI* (2013).
19. MacIntyre, B., Gandy, M., Dow, S., and Bolter, J. D. DART: a toolkit for rapid design exploration of augmented reality experiences. In *Proc. UIST* (2004).
20. Morris, M. R. Web on the Wall: Insights from a Multimodal Interaction Elicitation Study. In *Proc. ITS* (2012).
21. Morris, M. R., Danielescu, A., Drucker, S. M., Fisher, D., Lee, B., m. c. schraefel, and Wobbrock, J. O. Reducing Legacy Bias in Gesture Elicitation Studies. *Interactions 21*, 3 (2014).
22. Morris, M. R., Wobbrock, J. O., and Wilson, A. D. Understanding Users' Preferences for Surface Gestures. In *Proc. GI* (2010).
23. Nacenta, M. A., Kamber, Y., Qiang, Y., and Kristensson, P. O. Memorability of Pre-designed and User-defined Gesture Sets. In *Proc. CHI* (2013).
24. Nebeling, M., and Dey, A. K. XDBrowser: User-Defined Cross-Device Web Page Designs. In *Proc. CHI* (2016).
25. Nebeling, M., Huber, A., Ott, D., and Norrie, M. C. Web on the Wall Reloaded: Implementation, Replication and Refinement of User-Defined Interaction Sets. In *Proc. ITS* (2014).
26. Oh, U., and Findlater, L. The Challenges and Potential of End-User Gesture Customization. In *Proc. CHI* (2013).
27. Ouyang, T., and Li, Y. Bootstrapping personal gesture shortcuts with the wisdom of the crowd and handwriting recognition. In *Proc. CHI* (2012).
28. Piumsomboon, T., Clark, A. J., Billinghurst, M., and Cockburn, A. User-defined gestures for augmented reality. In *Proc. INTERACT* (2013).
29. Song, J., Sörös, G., Pece, F., Fanello, S. R., Izadi, S., Keskin, C., and Hilliges, O. In-air gestures around unmodified mobile devices. In *Proc. UIST* (2014).
30. Taranta II, E. M., Samiei, A., Maghoumi, M., Khaloo, P., Pittman, C. R., and LaViola Jr., J. J. Jackknife: A reliable recognizer with few samples and many modalities. In *Proc. CHI* (2017).
31. Vatavu, R. User-Defined Gestures for Free-Hand TV Control. In *Proc. EuroITV* (2012).
32. Wobbrock, J. O., Morris, M. R., and Wilson, A. D. User-Defined Gestures for Surface Computing. In *Proc. CHI* (2009).
33. Wobbrock, J. O., Wilson, A. D., and Li, Y. Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. In *Proc. UIST* (2007).