Instant User Interfaces: Repurposing Everyday Objects as Input Devices

Christian Corsten **Ignacio Avellino** **Max Möllers** **Jan Borchers**

RWTH Aachen University
52056 Aachen, Germany
{corsten, max, borchers}@cs.rwth-aachen.de, ignacio.avellino@rwth-aachen.de

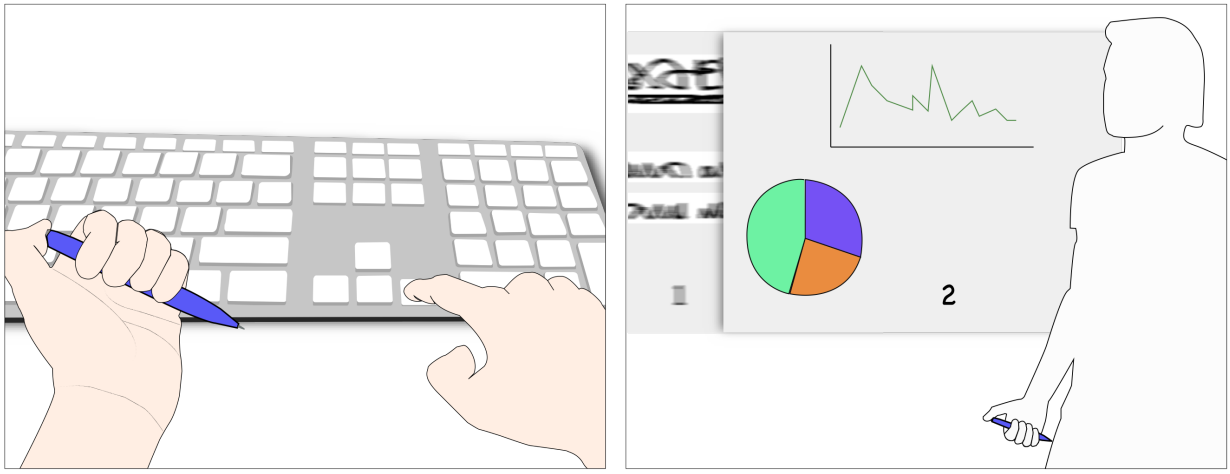

Figure 1. A presenter forgot to bring her presentation remote. (Left) She repurposes a pen as clicker and pairs it with the keyboard shortcut for advancing the slides by pushing both key and pen button simultaneously. (Right) The presenter advances the slides by pushing the pen — an almost identical substitute for the presentation remote.

ABSTRACT

Dedicated input devices are frequently used for system control. We present *Instant User Interfaces*, an interaction paradigm that loosens this dependency and allows operating a system even when its dedicated controller is unavailable. We implemented a reliable, marker-free object tracking system that enables users to assign semantic meaning to different poses or to touches in different areas. With this system, users can repurpose everyday objects and program them in an ad-hoc manner, using a GUI or by demonstration, as input devices. Users tested and ranked these methods alongside a Wizard-of-Oz speech interface. The testers did not show a clear preference as a group, but had individual preferences.

Author Keywords

Instant UIs; everyday objects; input devices; affordances

ACM Classification Keywords

H.5.2. Information Interfaces and Presentation: User Interfaces - Input devices and strategies.

ITS'13, October 6–9, 2013, St. Andrews, United Kingdom.

ACM 978-1-4503-2271-3/13/10...$15.00.
http://dx.doi.org/10.1145/2512349.2512799

INTRODUCTION

People often use dedicated input devices, such as a TV remote or light switch, to control technical systems. These controllers generally offer physical affordances [20] and tactile feedback via buttons or knobs, that support comfortable, eyes-free use even in, e.g., badly lit environments [29]. However, in situations in which the dedicated input device is missing or out of comfortable reach, this may provide an inconvenience, such as when a user always needs to get up from their bed to turn off the lights because the switch is next to the door.

If such dependency on a dedicated input device could be loosened by just grabbing an everyday object in the vicinity and repurposing it as a controller substitute, such breakdowns could be overcome in an ad-hoc manner. Imagine a presenter who forgot to bring her presentation remote. Without it, she would be rooted to the computer to control the presentation, limiting her freedom to move about and use expressive body language. To improvise, she grabs a pen, pushes its button while hitting the "Next" key on the keyboard to pair the control. From now on, she can press the pen button to advance the slides remotely (Figure 1). Despite being an improvised substitute, this "clicker" provides qualities similar to the original remote: The pen is unobtrusive and can be pushed blindly, providing a smooth presentation experience for both the audience and the presenter. We call such improvised and ubiquitous controllers *Instant User Interfaces*.

To let users repurpose and interact with everyday objects, we implemented a marker-free object tracking system that can determine an object pose as well as detect touches on the object surface, allowing users to give semantic value to their interaction with the object.

In summary, this paper has three major contributions:

1. The Instant UI concept, its interaction design, and benefits.
2. A technical solution for creating Instant UIs capable of recognizing and tracking unaltered everyday objects.
3. A study on three end-user programming methods for Instant UIs (speech, demonstration, GUI) to link actions on everyday objects to desired actions in an ad-hoc manner.

INSTANT USER INTERFACES

We define an *Instant UI* as a user interface that lets a user select a set of arbitrary physical objects within reach to **instant**iate it as controller for a technical system. Since the need for Instant UIs may arise spontaneously, mappings from object to system can be established **instant**ly, i.e., in an ad-hoc manner, by means of end-user programming.

Instant UIs are particularly related to three fields in HCI: Ubicomp [26], Organic UIs [15], and Tangible UIs (TUIs) [16]. Everyday objects are graspable, cover a wide range of natural shapes, and are — by definition — ubiquitous, such that Instant UIs "weave themselves into the fabric of everyday life" [26]. In comparison to Instant UIs, TUIs are designed to provide intentional physical affordances. Although not designed for their new use since they need to be instantly available, Instant UIs still target at exploiting both used and unused object affordances [20, 13] to map them to the system's digital or physical counterparts, such as dedicated buttons, knobs, or sliders (Figure 2). Physical affordances allow for comfortable, ergonomic, and eyes-free use. For example, eyes-free operation is beneficial in bad lighting conditions, when the visual channel is already occupied, for subtle interaction, or to lower cognitive and physical effort [29, 24].

The term *Physical Instantiation* coined by Ishii and Ullmer [16] originally refers to leveraging virtual widgets to the tangible world. Phidgets [6, 5], e.g., is a toolkit of physical widgets to rapidly develop physical interfaces that link back to virtual widgets, such as a physical button that is pushed to open a new e-mail message window. In Instant UIs, instantiation is also used to create a new physical instance from an existing one, like the pen that substitutes the presenter remote.

Basically, any artifact could become an Instant UI. Be it a plain object, a UI that is not an Instant UI (so far), or even an Instant UI that is turned into another Instant UI. Likewise, instantiation is open to different triggers. It could be: (1) user-triggered, e.g., when the user picks up the object, (2) time-triggered, i.e., the object is automatically turned into an Instant UI after a certain time of (non-)use, (3) proximity-triggered, i.e., an object becomes an Instant UI when it it close to another object, or (4) situation-triggered, e.g., when PowerPoint is running, the pen is automatically turned into a remote. In the remainder of this paper, we refer to user-triggered instantiation.

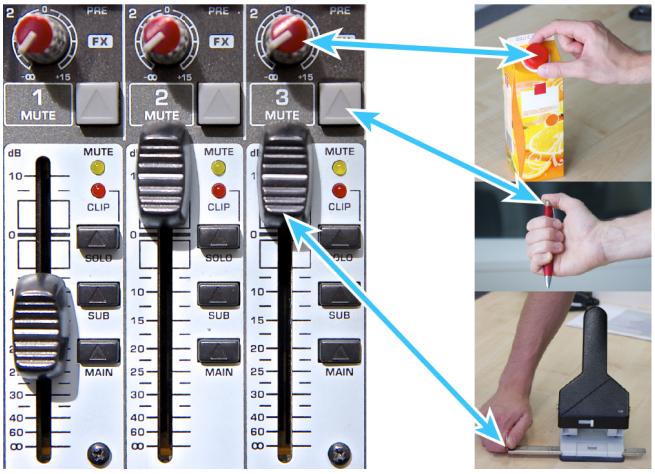

Figure 2. Everyday objects (right), such as a juice carton, pen, or hole puncher, provide controls with physical affordances (rotary knob, push button, slider) as used in dedicated input devices, such as a mixer (left).

Motivation and Benefits

Instant UIs provide several benefits. Here are some examples:

Improvisation. When a missing controller is needed urgently, the user can improvise by picking a nearby object as a substitute, such as a pen to advance presentation slides instead of using a remote control.

Convenience. Objects can serve as convenient mobile controllers for stationary input devices. A stapler on the desk, e.g., can be pushed to control the light instead of having to get up and reach for the light switch next to the door.

Usability. When a controller is too complicated to use (bad ergonomics, false affordances, or too many controls), Instant UIs can improve usability by outsourcing selected functions to objects with better ergonomics and simpler affordances, like a simplified TV remote with reduced functionality (channel +/- and volume control) but bigger buttons. Modal input could be substituted with space-multiplexed controls, which are faster to acquire and more comfortable [4].

Duplication. Instant UIs also allow us to create physical copies of existing controllers, such as additional joysticks for a multiplayer game console when friends drop by.

RELATED WORK

First, we will contrast Instant UIs to alternative approaches. Then, we will present related work on interaction with everyday objects and appropriate tracking technologies.

Instant UIs vs. Other Approaches

Smartphones with their dynamic touchscreen UIs already substitute various dedicated input devices. Samsung, e.g., provides an app [22] that allows users to control their TV via their smartphone. Since users generally have their smartphone with them, the substitute controller is in direct reach. However, due to their flat screen, smartphones lack physical affordances and tactile feedback. Even though the user knows the spatial UI layout of an app, incoming notifications may be touched accidentally and confound the input.

Speech interfaces also lack physical affordances and tactile feedback. Moreover, when quick closed-loop feedback is required, e.g., when dimming the lights, adjustments by speech may be tedious and slow. Background noise can interfere with speech recognition, and using spoken commands may not always be appropriate: During a presentation, speech commands not only feel awkward for the presenter, but may also annoy the audience.

Gesturing in mid-air is not always appropriate either. In a presentation, the audience would take notice of the presenter waving her arms in the air each time a new slide is about to show up, disrupting the presentation flow and annoying the audience. The presenter would need to avoid gestures that could trigger a command, increasing her cognitive load.

Interaction with Everyday Objects and Surfaces

Becoming independent of a dedicated physical controller can be achieved in two different ways: (1) by eliminating the physical controller *entirely* or (2) by letting the user repurpose *some* object as a physical controller substitute.

On-body interaction [10], such as Skinput [11] and Imaginary Phone [8] are a hybrid between (1) and (2). Users can touch their skin, e.g., the lower arm, to repurpose it as a touch device. Similarly, in WorldKit [28] users draw with their hands on any surface in the vicinity to define it as interactive touch area. Yet, these research projects do not build on affordances and lack the tactile feedback of physical knobs and buttons. Skinput and Imaginary Phone are independent of the user's visual attention, but require her to recall the spatial layout of the invisible interface from memory during interaction and thus may increase cognitive load. The same argument holds for other mid-air gesturing systems [7].

Opportunistic Controls (OCs) [13], Smarter Objects [14], and iCon [3] address option (2). For example, OCs exploit unused affordances of physical items in a mechanic's work space, such as grooves in a wiring harness or screws. Unlike Instant UIs, however, OCs use AR markers for tracking when a gesture intersects with an OC's physical geometry. Furthermore, they do not exploit depth information and only consider spatially fixed parts. Smarter Objects let users re-program physical controls from electronic devices, such as the tuner knob of a radio, to different functions, and iCon allows the user to use everyday objects to control background tasks of desktop applications, like a bottle that is turned to adjust the speaker volume. Only Smarter Objects and iCon provide end-user programming for linking the target interface with the physical object. While iCon focuses on desktop use, Smarter Objects let the user program the object with a tablet computer. However, both solutions build upon a visual UI as the only method for end-user programming, and iCon shifts the user's locus of attention away from the objects while programming.

Object Tracking and Manipulation

As we want to track everyday objects and detect interaction with them, we need to have a system that can be used for any shape of object and then interpret the data to find touches. Such a system should: (i) allow objects to be recognized quickly, (ii) allow recognition without altering the objects by, e.g., adding markers, which would require the user to constantly pay attention to not occlude the markers, and (iii) tolerate partial occlusion by the user's hand.

iCon [3] and DisplayObjects [1] require the user to augment objects with visual markers and thus cannot be used. Wear Ur World [19] and OmniTouch [9] require hardware mounted on the user's body. Despite mobile use of these systems, interaction with objects might not be natural due to the fact that the user is constrained by the hardware worn.

Since we do not want to modify the object, we need to sense the touches remotely. Here, we will focus on vision-based systems as they provide superior spatial resolution compared with other, e.g., ultra-sound-based systems. The simplest solution to scene understanding is to interpret the depth data as a point cloud. This is similar to how KinectFusion [17] works. It can reconstruct a 3D scene and segment it into connected point clouds. This way it can mimic physical behavior and have real objects interact with virtual objects, however, this solution is limited to object-agnostic interaction metaphors because it does not recognize objects. Even a simple shirt with interactive buttons is not possible. For such object-specific interpretations of user input, we need to recognize objects and thus turn the point cloud representation into an object representation.

Another alternative for tracking are template matching methods using 2D images such as [18], which require the system to be trained with different object poses in order to recognize its current pose in a scene. While these methods allow for characterizing changes in the object position and rotation, the movement granularity that the system is able to track would depend on the used image training set, and is usually low.

3D Puppetry [12] is a technical solution comparable to ours; the main differences lie in the object recognition phase. They use SIFT features which means that their objects cannot be smaller than a certain size (roughly 7 cm x 8 cm x 8 cm) and this process is susceptible to light condition changes. Our approach, which relies on geometric descriptors based on the 3D features of objects, is only limited by the resolution of the used sensor, does not limit the size of the recognized objects that strictly, and is unaffected by light conditions. Additionally, we provide an extensive qualitative analysis of the tracking and touch detection capabilities of our approach.

We now present our tracking approach. It only needs an initial model of the object to be tracked, which can be generated once before the system is initiated by a laser scan, providing the system with its shape and solid color for reliable tracking. Alternatively, digital 3D models could be provided by manufacturers, since they use them for fabrication anyway [25]. Our tracking approach also allows us to define different touch areas on the object that can be mapped to different events.

RECOGNITION, TRACKING, AND TOUCH DETECTION

First of all, we need to define the context of our system: using a single camera. This increases applicability of the system due to its very simple setup. However, this means we will not be able to determine the orientation of objects that appear rotation invariant from the current viewpoint, e.g., if the handle

of a mug is hidden behind the mug itself, we cannot accurately determine its rotation. Similarly, touches can only be recognized when they are in front and not behind the object w.r.t. the observing camera (see limitations).

In order to recognize an object, i.e., find it in a scene, we need to first have a digital representation of it. A base option is to store the object as a 3D mesh or a point cloud. This can be created digitally—most physical objects that surround us start out as a CAD model—it can come from laser scans of an actual object, or we can use a depth-camera to create the model [18]. We chose to laser scan our artifacts as this gives the highest accuracy without relying on available CAD models.

Before going into details, we first sketch the overall algorithm (Figure 3): (1.) Raw data is acquired from the depth camera. (2.) We create a point cloud from the depth data by unprojecting the pixel-based information into 3D space. (3.) We (optionally) remove the background plane to reduce the number of points to process by allowing the user to click three on-screen points during system setup that compose the background plane. (4.) We use our object recognition algorithm to get an initial pose estimation. (5.) The object is tracked from one frame to the next and its position and orientation are updated. (6.) If a finger is in front of the object, we detect touches by depth-thresholding.

We now go through each step to explain the concept of the algorithm. Step 1 consists of receiving color and depth data from the Kinect, and in step 2 we generate a point cloud of those raw images. In step 3, the user selects three points on the background. This defines a plane and we can remove points in the vicinity to the plane. This allows us to remove a lot of the points if, e.g., the object is standing on a table, allowing faster completion of step 4, the initial pose estimation. Thus, step 3 is only done once as part of the setup.

Initial Pose Estimation

In this step, we match our object with the depth data we receive from our camera. Yet, every object is only partially visible for a single camera. Additionally, objects will be occluded by themselves or by the user interacting with them. This means that one needs to find an object recognition method that is robust against occlusions. For example, global features are likely to be occluded so we selected local features. Additionally, (our) depth cameras provide rather noisy images. One could, e.g., average succeeding images over time, as KinectFusion [17] does, but this might result in lag, as information from previous images is taken into account when calculating the current state. Instead, we use curvatures between medium-distant point pairs as a local feature descriptor.

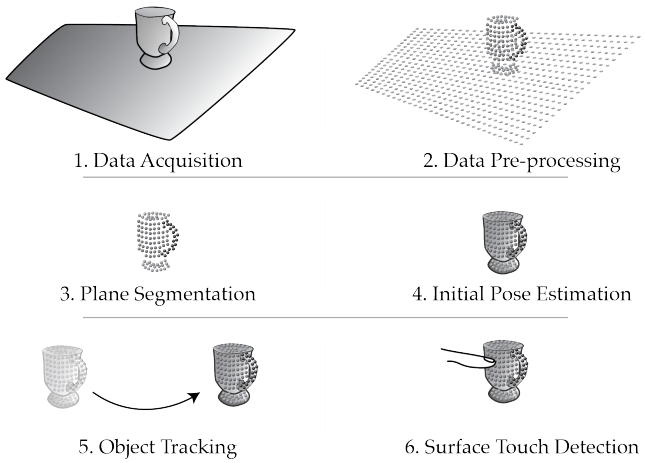

Figure 3. Tracking and touch detection algorithm. (4.) is done only once.

For the object recognition part, we use a slightly modified version of the work from [21], an object recognition algorithm that is tailored to noisy single-view scan data, similar to our problem domain. This initial pose estimation is quite robust, yet it needs several seconds to find the object in the point cloud on our machine (MacPro, 2.26 GHz, 6GB RAM). As this is too slow for real-time tracking of the object, we will employ another algorithm for tracking the object in step 5.

Object Tracking

After having found an object, it is very unlikely for the object to disappear or to be moved very far from one frame to the next, given high frame rate cameras and realistic object movement. This gives us two advantages: (i) we can limit the search volume for the object in the next frame by creating a bounding box around the last known position, and (ii) instead of running the full object recognition every frame, we just track the object using Iterative Closest Point (ICP) [2]. ICP is an algorithm that maps two point clouds (a model and a template) to each other while minimizing point distances in each step until a convergence criterion (here distance) is reached (Figure 4.5). It then returns the mapping that moves and rotates the template onto the model.

Before applying ICP and updating our pose estimation, we need to prepare our data to be usable for ICP (Figure 4): (1) We start with knowledge of where the object was in the last frame, i.e., a rotation and translation that maps our stored model into the 3D world coordinates of the camera system. For the first frame we use the object recognition mentioned in the last section. (2) The object moves. (3) We now get new input from the camera, transform it into a point cloud, and create a bounding box around the previous location and assume that the object didn't move too far from it. (4a) We remove the background plane, use color filtering to remove hands that might occlude the object based on the solid color of the object, and crop the points outside of the bounding box. We now have one of the two point clouds for the ICP: the model. (4b) As we know where the object was in the last frame, we can place our model in the 3D space and take the points that face towards the camera. This point cloud is our expected input: *the template*. Without object movement (and noise in the data) the template would be exactly the same as the model. We only use the front-facing points for the template because the model is also only seen from one side and the ICP will not give good results if we require it to map the object front side to a full object. (5) We now apply ICP, which returns a mapping from the template to the model. (6) We can now store this information and update our pose estimation.

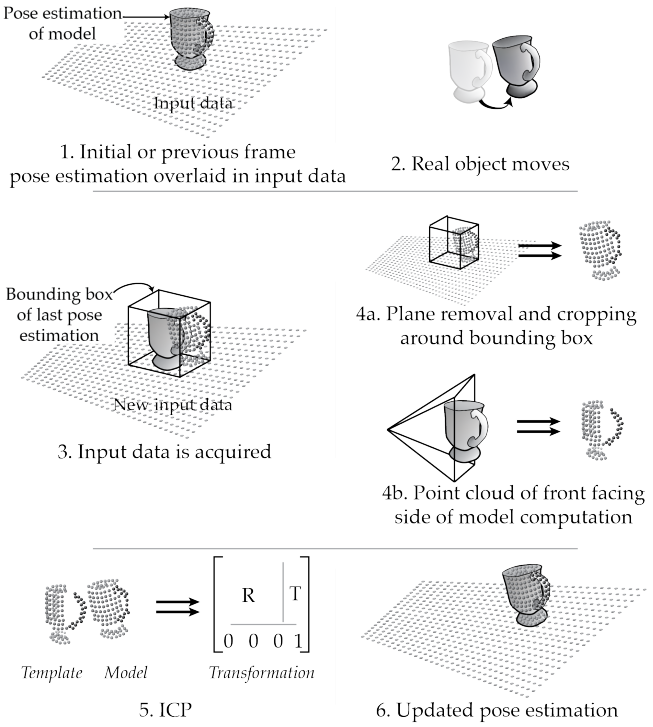

Figure 4. Steps for updating the pose estimation from frame to frame.

Now we know where the object is and how it is oriented. This enables us to check if and where fingers are touching the object.

Touch Input and Touch Occlusion

As mentioned, we use color filtering to differentiate between objects and the hand. As color values in the standard RGB model vary a lot under different lighting conditions even for the same object, we use Hue, Saturation, and Value (HSV) instead. This makes the thresholding more robust to different ambient lighting and especially self-shadowing of objects.

Before going into detail, we explain the challenges in calculating touch locations in real time. A finger touching the object means that the object model touches or intersects the finger point cloud. Even if we had a full model of the finger (which we do not) we would need to check both point clouds for per-point "intersections" to find the touch area. Unfortunately, this would require significant computation.

Another problem is that the finger occludes the object. Although our object recognition gives us the pose of the object, allowing us to restore which points of the model are occluded by the finger, we do not know whether the finger is touching the object or whether it is just in front of it. Yet, if we assume a static finger thickness, we can check for the touch by comparing depth of the front-side of the finger plus finger thickness to the depth value of the object's front side. To be more specific, we need to check this for every point of the finger with the point lying "behind" it. Calculating the point behind another, however, would require us to cast a ray (parallel to the camera axis) through that point and intersect it with every triangle of the model, which would also require a lot of computation.

Our approach simplifies the 3D calculations to 2D image comparisons and is very fast as it exploits the capabilities of modern graphics cards: instead of casting rays, we render the object as it would be seen from the perspective of the camera. Theoretically, this is also quite an expensive computation, but modern GPUs are highly optimized for this task so that they can deliver this within $5ms$, making it a very feasible approach. We also project the points of the finger into a 2D image with the same viewport as the other image. This means that we can now compare pixel by pixel whether the finger is in front of the object. As we store the depth data of the model and finger as well, we can thus use depth thresholding for our touch detection.

This allows us to identify touches on moving objects, eliminating the need to know the distance from the sensor to the touched object a priori as required by Wilson et al. [27], where a "snapshot" of the depth image is taken when the touch surface is empty. Our approach is similar to OmniTouch [9] where the depth of the fingertip is compared to the depth of the object behind it in order to identify touches.

The touch detection process is now explained in Figure 5: (1) We start with the point cloud data and the current object pose estimation. (2) We segment the finger points by color filtering. (3) We now treat the data as 2D image as seen from the camera to ease computation and hit detection in the next steps. (4) We compute the contour by calculating the convex hull of the finger points. (5) We take the point farthest away from the center of the hull as the fingertip. (6) We render the model and project the fingerprints from the same viewport. (7) We compare points in the vicinity (3 pixels) of the fingertip with the model image and perform depth thresholding to decide whether there is a touch or a hover.

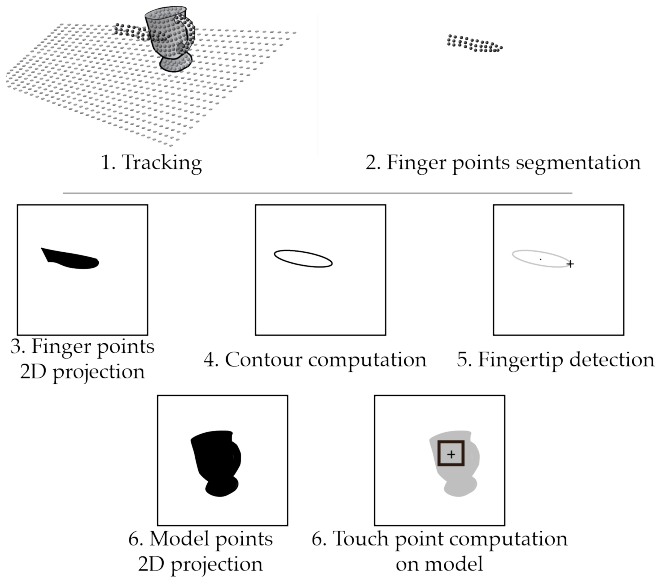

Figure 5. Steps for detecting touches on arbitrary objects.

We now know whether we hit the object. In order to know where we hit the object, we initially define different touch regions on the model by marking the respective areas in the model or point cloud with a specific ID. Currently, we store this in the color value that is not used otherwise. This ID can then be checked in the last step of Figure 5 and will tell us which touch zone we hit.

Evaluation of Tracking

Our system has two main parts, the tracking of the object and the detection of touches on it, and we will evaluate them separately. The tracking will be measured in a controlled environment without user interaction to only capture system and not user or task performance. Additionally, we will show results from a real application scenario to see how reliable the system is when the user is frequently occluding parts of the object. We will also show how well touches are recognized.

The evaluation was done on a MacPro with 2.26 GHz, 6GB RAM and an Nvidia GTX 580 graphics card. Depth data is generated from a first-generation Microsoft Kinect that returns VGA resolution RGB and depth data. The latter are 11 bit and have a base accuracy of about 2 mm at 50 cm depth and get worse farther away.

Controlled Environment

The system was able to track objects with 7-10 frames per second, i.e., it takes 100-140 ms to process the incoming frame and update the pose of the object. The touch detection that follows afterwards does not increase the performance to any measurable degree. The tracking performance varies depending on (a) the number of points of the model and (b) the number of points of the template. We already do a subsampling, i.e., only take every 16th point, after calculating the front-facing points of the model, but did not do this for the point cloud from the Kinect camera to not lose accuracy. For the above results, the sub-sampled model consisted of about 600 points, and the points extracted from the Kinect image were about 3500. This also means that objects closer to the object camera generate larger point clouds than objects farther away, leading to differences in performance. In the future, we will take a look at ways to adapt to those factors and get a more flexible trade-off of processing time vs. accuracy.

To evaluate system accuracy of the system, compared it to a baseline VICON motion tracking system with sub-millimeter accuracy. We looked at position and orientation error. For a more controlled environment, we put the objects on a Lego toy train with a circular track layout. We used one object with few features (a mug) and one with many features (a toy duck). We put them on the toy train in varying starting rotations and let them do several circles in both directions.

Position error in a space of up to 122 cm of depth is reported in Table 1. As we can see, the biggest error is in the depth dimension, partly due to the Kinect's own limited depth resolution (around 4 mm error in this space). Errors in the width and height dimension are comparatively small. Altogether, this makes the system very usable for the envisioned use case.

The rotation error is measured as the norm of the difference of rotation quaternions. The norm is in the range $[0, \sqrt{2}]$, where 0 equals identity and $\sqrt{2}$ equals a rotation in the exact opposite direction. We use the norm of quaternions as they are a good representation to compare.

Rotation tracking works as expected: as soon as an asymmetric feature is occluded, the system cannot track the rotation correctly and can only exclude states where that feature would be visible. Fortunately, in most cases the system is able to catch up as soon as the feature is visible again, i.e., we see the handle again and the system reports the correct orientation. Multiple cameras would solve this issue. However, as long as the handle is visible, we see an average of 0.0634 with standard deviation of 0.0201. This average norm is similar to a rotation of 7.23° along a single axis.

	Mug	Duck
Width:	16.27 ± 12.51	15.22 ± 13.57
Depth:	24.04 ± 15.65	28.73 ±17.70
Height:	4.32 ± 3.06	4.11 ± 3.8

Table 1. Tracking error of the position in mm ($M \pm SD$). Even the depth error is considerably small.

Limitations of the System

If the system loses track of an object due to full occlusion or very fast movement, it can only recover if the object is moved back into the region of its last position. Instead, we would then start the full recognition process over again to find it.

As of now, the recognition algorithm compares against a database of multiple objects, yet the tracking part is implemented for one object at a time. As the current code does not exploit parallelization, tracking of a maximum of objects similar to the number of CPU cores in the systems should be possible with similar performance. Similarly, GPU optimizations could be considered.

The system is based on a database of models with a static point cloud or mesh. It would be interesting to see how we could support deformable shapes. The question how to define and store the "deformability" of an object is a challenge by itself. It could lead to insight on how to recognize those shapes as the standard features used to recognize objects are based on a fixed shape: we, e.g., use point-pair relations such as distance and curvature in the neighborhood.

USER STUDY: INTERACTING WITH INSTANT UIs

After evaluating the system performance in a controlled setting, we now move on to real tasks and observe how users experience the new interaction metaphor of Instant UIs: We evaluate three different ways of end-user programming for everyday objects, i.e., using speech, by demonstration, and with a GUI, and observed system performance w.r.t. tracking failures and touch detection accuracy in these applications scenarios.

Scenarios

Scenario *A* represents our introductory example: Navigating presentation slides using a clickable pen. Scenario *B* asked

the user to operate a dimmable light source using a mug. We chose these (*A*, *B*) as they represent two different applications fields for Instant UIs (improvisation, convenience) and deal with two different types of control for input (discrete, continuous).

End-User Programming Techniques

In the first step, users had to establish mappings from object (*A*: pen, *B*: mug) to the set of needed actions from the target interface (*A*: next/previous slide, *B*: brighten/dim light).

Speech. Using speech, a user simply told the system how to establish mappings from object to target interface. She was not limited in using specific commands/vocabulary. She could have said, e.g.: "When I push the pen button once, go to the next slide.". Based on the user's commands, the experimenter established the mappings manually (Wizard-of-Oz).

Demonstration. The user established a mapping by performing a gesture on the object while simultaneously controlling the target interface with an existing physical controller. Pushing the pen button while pressing the right arrow-key on the computer keyboard, e.g., paired those two events and triggered the action mapped to the keyboard shortcut (next slide) each time when the pen button was pushed again. Hence, slides were advanced by pen click, since this was mapped to the keyboard shortcut that triggered the next slide.

Graphical User Interface. A dedicated GUI (Figure 6) enabled the user to establish mappings between object and target interface by pointing with the object at it. The left side of the GUI lists physical components of the object that has an affordance (e.g., the pen button), the right side lists the action needed for the tasks from the target interface (e.g., next slide). Moving the object in mid-air in front of the GUI display controlled a virtual cursor, and hovering over a GUI element for at least two seconds selected it. Mappings were established by selecting one element from the left and one from the right side. A line indicated the pairing. Old mappings were overwritten. Using the object as a pointing device has the benefit that the user does not need to tell the system which object she wants to program since the system will identify the object in use.

We have chosen these methods to represent diversity: speech and GUI are a virtual-to-physical and -virtual, whereas demonstration is a physical-to-physical mapping technique. Additionally, speech is probably the most natural, demonstration the most practical, and the GUI the most typical approach [3, 14] for everyday object programming.

Participants, Procedure, and Setup

We recruited 12 participants (4 females), aged 19–28 (*M = 24.33*, *SD = 2.50*) to participate in our study. A session lasted 45 minutes. First, the user was introduced to our system and the Instant UI concept including the three end-user programming methods. Next, the user made herself familiar with the first scenario presented by performing the demanded tasks (Table 2) using the dedicated physical controller (*A*: computer keyboard, *B*: rotary knob). Subsequently, for each end-user programming condition, the user was asked to program the

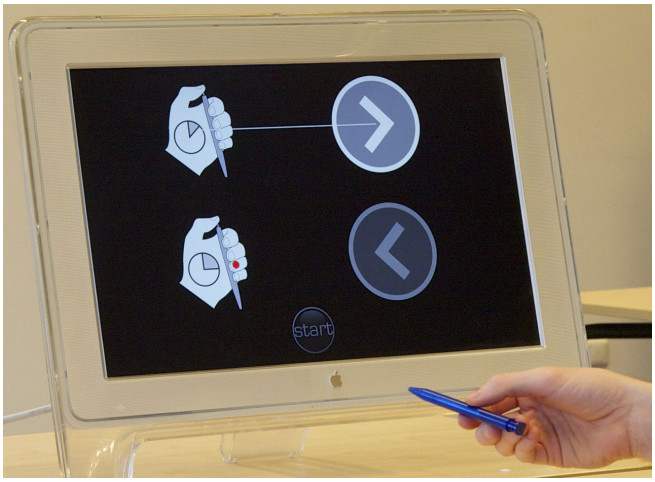

Figure 6. GUI for programming the pen as a remote. Pointing with the object at the GUI controls a red cursor with which the user selects a pen gesture visualized on the left side of the GUI (push, push-and-hold) and maps it to an action from the target interface (right side: next slide, previous slide). Established mappings are characterized by a line. Programming is finished by selecting the "start" button at the bottom.

object (*A*: pen, *B*: mug). For the pen, the user could choose from a push and a push-and-hold gesture and map those to the commands "next slide" and "previous slide". We started measuring a push as soon as the user started to touch the pen button. The system recognized a push when the user touched the top of the pen for 400 ms to 1200 ms, and a push-and-hold when this happened for 1200 ms to 1700 ms. Push-and-hold events were recognized consecutively, and their associated action fired every 500 ms if the user kept touching the top of the pen. For the mug, the user could choose between the handle facing the user and it facing away from the user and map these positions to "light is at maximum brightness" and "light is turned off". After each programming phase, the user was asked to perform the tasks (Table 2) only by using the programmed object. Having performed all three conditions, the scenario was concluded with a questionnaire. The remaining scenario was tested accordingly. Both the order of the scenarios and end-user programming conditions were randomized. Figure 7 shows a complete setup of the system and the study. To ease implementation, the dimmable light was emulated on a screen. For the physical knob, we used a USB-wired Shuttle Xpress Jog Knob from Contour Design.

Objects and gestures for the scenarios (Table 2) were based on the results of a preliminary study in which participants were asked to grab an everyday object out of 63 ones available that they considered a suitable controller substitute for *A* resp. *B*.

Results and Discussion

Users rated their preference for the three end-user programming methods on a 9-point Likert scale (1 = definitely not preferred, 9 = definitely preferred, Figure 8). For the presentation scenario, the data shows that participants rated programming using the GUI best (*M = 5.25*), followed by programming by demonstration (*M = 4.92*) and using speech (*M = 4.83*). However, there was no significant effect of the mapping method on the preference rating ($\chi^2_{2,N=12} = 0.30,\ p =$

Scenario	A: Presentation	B: Light
Object	Clickable Pen	Mug
Gestures on the object	push, push-and-hold	turn clockwise, turn counter-clockwise
Actions on the target	next slide, previous slide	dimming, brightening
Precondition	The presentation has been started.	The brightness is at maximum.
Tasks	1. Go to the next slide (three times) 2. Go to the previous slide. 3. Jump two slides forward. 4. Jump three slides backward.	1. Dim it to ca. medium. 2. Dim it to ca. minimum (keeping it on). 3. Brighten it to ca. medium. 4. Brighten it to maximum. 5. Turn it off

Table 2. Tasks for the users to execute per scenario after programming.

Figure 7. Setup of the system for the light scenario (left to right): Kinect, display, mug, rotary knob. The user is programming the mug by demonstration: mug and rotary knob used to control the light (on screen) are rotated at the same time. From now on, mug rotation dims the light.

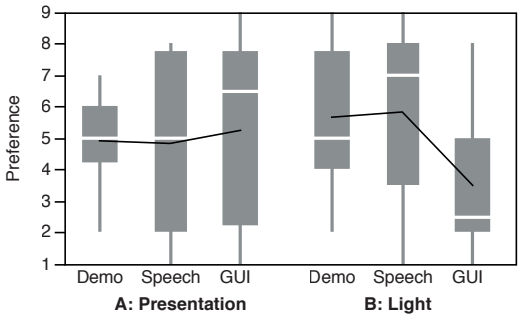

Figure 8. Box plot showing aggregated results for the users' preference on end-user programming methods. Black lines connect the means of each condition. (Left) Results for the presentation scenario showed no clear preference for a method. (Right) Programming by using the GUI was rated worse than the other methods for the light scenario because pointing with the mug was tedious because of its weight (430 g).

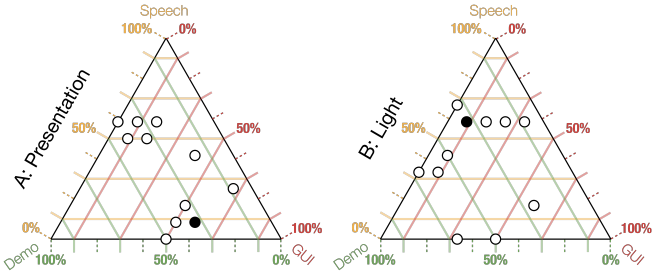

Figure 9. Users' preference on end-user programming methods. Ternary plots are used to visualize data points of three variables in planar space. Each data point represents the percentages of preference for each method and adds up to 100%. The closer a data point is to a corner, the more the corresponding method is preferred over others. (Left) Results for the presentation scenario show no clear preference overall since data points are accumulated towards the center. (Right) Users rated against the GUI method. Filled circles indicate two data points.

.859). Taking a closer look at the individual ratings shows that participants had diverse preferences (Figure 9, left). Interestingly, although many users rated the GUI or the speech interface strongly negatively (data points are far away from these corners), ratings for demonstration were less extreme.

In the light scenario, speech was rated the best interface (*M = 5.83*) closely followed by programming by demonstration (*M = 5.67*). The GUI, however, was rated worse (*M = 3.50*). Again, there was no significant effect of the mapping method to the preference rating ($\chi^2_{2,N=12} = 3.87,\ p = .144$). Concerning the individual rankings, participants had diverse preferences, but collectively agreed on rating against the GUI approach (Figure 9, right).

Programming the pen in all three conditions was considered a simple approach: "*It was simple – no complications.*". Programming the mug, however, was considered more difficult for demonstration and the GUI. Some participants found it difficult to perform synchronous rotation with both hands. They had to rotate the mug on a traverse plane, while rotating the rotary knob on a frontal plane. This setup was chosen for the study since a light dimmer is mounted onto a wall (frontal plane), whereas a mug resides on a table (transversal plane). In fact, synchronously performing tasks on different planes of continuous motion is suboptimal [23]. Synchronized pushes (pen and keyboard), however, did not cause any difficulties for the users. Hence, programming by demonstration for continuous control was more difficult than for discrete control. Pointing with the mug was considered too tedious because of its weight (430 g for the mug vs. 6 g for the pen): "*The mug became heavy after a while*".

Touch Accuracy

For evaluating touch recognition accuracy, we recorded 511 touches during the tasks that involved clicking the top of the pen by manually annotating the video recordings of the studies. A first analysis of the collected data shows that in 66.91% of the time pushes on the pen button were successfully recognized (182 out of 272) and in 55.65% of all times the push-and-hold gesture was recognized successfully (133 out of 239) (Figure 10). The apparent low recognition rate of the touch detection comes from the time threshold used to discern between noise, push, and push-and-hold gestures. Eight users reported having problems with the system differentiating between push-and-hold and a push event, and as can be seen from the data collected, although the recognition rate seems low, false recognitions happened most of the time because either a push-and-hold event was mistaken for a push event (26.78%) or it was considered noise (13.39%)—since it

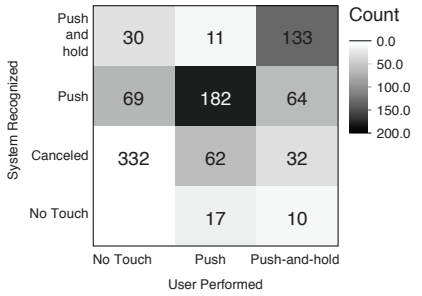

Figure 10. Touch recognition heat map for conditions with the clickable pen. The amount of touches performed by the users are shown in segments according to how the system recognized them. 332 correctly filtered out noise data points are not considered for the coloring of the heat map although this total is shown.

was not long enough to be a push event. Likewise for push events, most of the time they were not recognized successfully because they were either too short (22.79%) or considered to be push-and-hold events (4.05%).

From post analysis of video data, it can be observed how users did not respect the defined times for push and push-and-hold gestures, although they were instructed on how the system worked. This leads to misinterpretation between noise, push, and push-and-hold gestures (Figure 10). This is why we also report aggregated touch data ignoring the threshold for such gestures. If we consider both types of gesture (push and push-and-hold) to be simply a performed touch, and consider the detected type of gesture (push, push-and-hold or canceled for being too short) also simply as a detected touch, touch accuracy is as high as 80%, with 4.43% of the touches not recognized by the system and 16.23% of the time noise mistaken for a touch.

LIMITATIONS, FUTURE VISION, AND CHALLENGES

Our system exemplifies how Instant UIs can be implemented technically. Yet, relying on visual input limits the interaction to the field of view of the sensor(s) (a touch cannot be recognized on the back of an object), rotation detection is limited when using objects that appear rotation invariant from the perspective of the sensor, and touch recognition accuracy on moving targets is highly affected by noise introduced by the sensor. Also, the time needed for initial pose recognition and the requirement of setting up a tracking system limits its ad-hoc quality. In the near future, however, many spaces may be equipped with ambient tracking technology including directed projectors and speakers as envisioned in [30] to detect object interaction and provide visual and audible feedback, e.g., to give feedback on programmed mappings.

Besides such "intelligent" rooms, residing objects also need to become "smart". Sterling [25] envisions each object — be it a bottle or a light switch — to have a digital descriptor that contains meta data such as a 3D model of the object, available physical controls, an API descriptor of accessible functions (e.g., light on/off), etc. This vision is not unlikely, as such data is already available for the production of an object. Manufacturers could then add this data by simply putting RFID stickers to the objects.

Besides technical challenges, interaction design also poses questions to be solved, in particular how to solve the ambiguity of repurposed use and regular object use. A stapler, e.g., is intended to staple paper, but pushing its lever could also turn the lights on/off. To resolve this ambiguity, the user could convey her intention to the system by executing a specific gesture prior to the appropriated use. Taking a closer look at the exact gesture (sequence) might also work: To use the stapler as a button, a gentle touch with one finger could suffice to lower the lever. Actual stapling requires more force, e.g., pressing the lever using the entire palm.

Similarly, end-user programming poses further challenges. How can the user *enter* programming mode? Again, an explicit gesture could enter the mode, or the system detects that the user manipulates the object in a way that does not match the typical use of the object. Programming by demonstration needs further investigation as regards accuracy of programming. Users may perform slower and less accurate gestures than desired. For example, when rotating the mug while turning the light knob, the user may want to have a 1:1 mapping concerning the position of both knobs, but she may involuntary create an offset. System-corrected mappings are one solution to explore. Moreover, we will extend our evaluation on end-user programming such that the user always *demonstrates* an action before defining its result using demonstration, speech, or the GUI. This way, speech and GUI do not rely on the existence of a known set of actions that can be applied to an object.

SUMMARY AND CONCLUSION

We presented *Instant User Interfaces*, a new interaction metaphor that frees users from dedicated input devices, while maintaining the benefits of physical affordances and tactile feedback. In case a device is missing or out of reach, the user can improvise by grabbing an everyday object in the vicinity and repurpose its physical controls, such as the button of a pen, for input. To explore this idea, we implemented a marker-free object tracking system, with approx. $30mm$ position and $7°$orientation error, that senses touches on the object with more than 80% accuracy. Furthermore, the system allows to assign actions to changes in the tracked object pose and to touches on its surface. In a study, twelve users tested two Instant UIs: they navigated slides by clicking a pen and dimmed the lights by rotating a mug. To make those objects interactive, the users tested three different methods: By using a Wizard-of-Oz speech interface, a GUI, or manual demonstration, the users linked the physical actions for the everyday object (e.g., pushing the pen button) to the desired actions of the interface to be controlled (e.g., advance the slide). As a group, the users did not show a clear preference towards any method, but each of them had distinct individual preferences.

From the perspective of interaction design, most users agreed that everyday objects could be used as substitutes for dedicated input devices. Today, however, ambient tracking technology is not prevalent in the world. Our system, despite limitations in tracking speed, has provided first insights into Instant UIs. As tracking technology improves, it may become

ubiquitous and thus make Instant UIs an enticing solution to free users from dedicated input devices.

ACKNOWLEDGMENTS

We thank Marty Pye and Kashyap Todi for their support. This work was funded in part by the German B-IT Foundation and the German National Science Foundation (DFG).

REFERENCES

1. Akaoka, E., Ginn, T., and Vertegaal, R. DisplayObjects: Prototyping Functional Physical Interfaces on 3D Styrofoam, Paper, or Cardboard Models. In *Proc. TEI '10*, 49–56.
2. Besl, P. J., and McKay, N. D. Method for Registration of 3-D Shapes. In *Proc. TPAMI '92*, 586–606.
3. Cheng, K.-Y., Liang, R.-H., Chen, B.-Y., Laing, R.-H., and Kuo, S.-Y. iCon: Utilizing Everyday Objects as Additional, Auxiliary and Instant Tabletop Controllers. In *Proc. CHI '10*, 1155–1164.
4. Fitzmaurice, G. W., and Buxton, W. An Empirical Evaluation of Graspable User Interfaces: Towards Specialized, Space-Multiplexed Input. In *Proc. CHI '97*, 43–50.
5. Greenberg, S., and Boyle, M. Customizable Physical Interfaces for Interacting with Conventional Applications. In *Proc. UIST '02*, 31–40.
6. Greenberg, S., and Fitchett, C. Phidgets: Easy Development of Physical Interfaces Through Physical Widgets. In *Proc. UIST '01*, 209–218.
7. Gustafson, S., Bierwirth, D., and Baudisch, P. Imaginary Interfaces: Spatial Interaction with Empty Hands and without Visual Feedback. In *Proc. UIST '10*, 927–930.
8. Gustafson, S., Holz, C., and Baudisch, P. Imaginary Phone: Learning Imaginary Interfaces by Transferring Spatial Memory from a Familiar Device. In *Proc. UIST '11*, 283–292.
9. Harrison, C., Benko, H., and Wilson, A. D. OmniTouch: Wearable Multitouch Interaction Everywhere. In *Proc. UIST '11*, 441–450.
10. Harrison, C., Ramamurthy, S., and Hudson, S. E. On-Body Interaction: Armed and Dangerous. In *Proc. TEI '12*, 69–76.
11. Harrison, C., Tan, D., and Morris, D. Skinput: Appropriating the Body as an Input Surface. In *Proc. CHI '10*, 453–462.
12. Held, R., Gupta, A., Curless, B., and Agrawala, M. 3D Puppetry: A Kinect-Based Interface For 3D Animation. In *Proc. UIST '12*, 423–434.
13. Henderson, S. J., and Feiner, S. Opportunistic Controls: Leveraging Natural Affordances as Tangible User Interfaces for Augmented Reality. In *Proc. VRST '08*, 211–218.
14. Heun, V., Kasahara, S., and Maes, P. Smarter Objects: Using AR Technology to Program Physical Objects and Their Interactions. In *CHI EA '13*, 961–966.
15. Holman, D., and Vertegaal, R. Organic User Interfaces. *Communications of the ACM 51*, 6 (June 2008), 48–55.
16. Ishii, H., and Ullmer, B. Tangible Bits: Towards Seamless Interfaces between People, Bits, and Atoms. In *Proc. CHI '97*, 234–241.
17. Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., and Fitzgibbon, A. KinectFusion: Real-Time 3D Reconstruction and Interaction Using a Moving Depth Camera. In *Proc. UIST '11*, 559–568.
18. Liu, K., Kaleas, D., and Ruuspakka, R. Prototyping Interaction with Everyday Artifacts: Training and Recognizing 3D Objects via Kinects. In *Proc. TEI '12*, 241–244.
19. Mistry, P., and Maes, P. WUW - Wear Ur World - A Wearable Gestural Interface. In *CHI EA '09*, 4111–4116.
20. Norman, D. A. *The Design of Everyday Things*. Basic Books (AZ), 2002.
21. Papazov, C., and Burschka, D. An Efficient RANSAC for 3D Object Recognition in Noisy and Occluded Scenes. In *Proc. ACCV '10*, 135–148.
22. Samsung Electronics. Samsung SmartView. `http://www.samsung.com/au/tv/my-hub/apps-store/smart-view.html`. [Online; accessed 12-Aug-2013].
23. Serrien, D. J., Bogaerts, H., Suy, E., and Swinnen, S. P. The Identification of Coordination Constraints Across Planes of Motion. *Experimental Brain Research 128*, 1-2 (Sept. 1999), 250–255.
24. Shahrokni, A., Jenaro, J., Gustafsson, T., Vinnberg, A., Sandsjö, J., and Fjeld, M. One-Dimensional Force Feedback Slider: Going from an Analogue to a Digital Platform. In *Proc. NordiCHI '06*, 453–456.
25. Sterling, B. *Shaping Things*. MIT Press (MA), 2005.
26. Weiser, M. The Computer for the 21st Century. *Pervasive Computing, IEEE 1*, 1 (2002), 19–25.
27. Wilson, A. Using a Depth Camera as a Touch Sensor. In *Proc. TABLETOP '10*, 69–72.
28. Xiao, R., Harrison, C., and Hudson, S. E. WorldKit: Rapid and Easy Creation of Ad-Hoc Interactive Applications on Everyday Surfaces. In *Proc. CHI '13*, 879–888.
29. Yi, B., Cao, X., Fjeld, M., and Zhao, S. Exploring User Motivations for Eyes-Free Interaction on Mobile Devices. In *Proc. CHI '12*, 2789–2792.
30. Zerroug, A., Cassinelli, A., and Ishikawa, M. Invoked Computing: Spatial Audio and Video AR Invoked Through Miming. In *Proc. VRIC '11*, 31–32.