Custom-made Tangible Interfaces with TouchTokens

Caroline Appert, Emmanuel Pietriga, Eléonore Bartenlian, Rafael Morales González

Custom-made Tangible Interfaces with TouchTokens

Caroline Appert, Emmanuel Pietriga, Éléonore Bartenlian, Rafael Morales González
Univ. Paris-Sud, CNRS, INRIA, Université Paris-Saclay
Orsay, France
appert@lri.fr,emmanuel.pietriga@inria.fr,eleonore.bartenlian@u-psud.fr,morales@lri.fr

ABSTRACT

TouchTokens were introduced recently as a means to design low-cost tangible interfaces. The technique consists in recognizing multi-touch patterns associated with specific tokens, and works on any touch-sensitive surface, with passive tokens that can be made out of any material. TouchTokens have so far been limited to a few basic geometrical shapes only, which puts a significant practical limit to how tailored token sets can be. In this article, we introduce *TouchTokenBuilder* and *TouchTokenTracker* that, taken together, aim at facilitating the development of tailor-made tangible interfaces. *TouchTokenBuilder* is an application that assists interface designers in creating token sets using a simple direct-manipulation interface. *TouchTokenTracker* is a library that enables tracking the tokens' full geometry. We report on experiments with those tools, showing the strengths and limitations of tangible interfaces with passive tokens.

CCS CONCEPTS

• **Human-centered computing → Interface design prototyping**; *Gestural input*;

KEYWORDS

Multi-touch surfaces, Tangible Interaction, Customization

ACM Reference Format:
Caroline Appert, Emmanuel Pietriga, Éléonore Bartenlian, Rafael Morales González. 2018. Custom-made Tangible Interfaces with TouchTokens. In *AVI '18: 2018 International Conference on Advanced Visual Interfaces, AVI '18, May 29-June 1, 2018, Castiglione della Pescaia, Italy*. ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/10.1145/3206505.3206509

1 INTRODUCTION

Tangible interfaces have been designed for use in various domains such as music composition [14], storytelling [23], games [3, 26], teaching [21], programming [5, 11] and database querying [13, 25]. All of these interfaces feature physical tokens that aim at resembling actual objects from the targeted application area. Variations in the shape, size and material of these tokens all play an important role in providing the right manipulation affordances and conveying the proper semantics [22, 25]. Promoting tangible interfaces thus requires enabling designers to easily build tailor-made tokens that suit their specific needs.

The physicality of tangible interfaces makes them resistant to customization, however [12]. Various approaches to building tangible interfaces exist, such as vision-based frame analysis for diffused illumination tabletops (*e.g.*, [13, 14]), conductive tokens tracked on a capacitive surface (*e.g.*, [16, 27]) or specific sensors (magnetometers, Hall-effect sensors) augmenting the display surface in order to detect magnetic tokens [12, 18]. But whichever the technology considered, building and tracking tangible tokens remains an effortful process in terms of fabrication, software development, or both.

TouchTokens [19] offer an alternative solution, enabling the design of low-cost tangible interfaces. The general principle consists of designing tokens of varying shapes, all featuring notches that constrain how users grasp them. When users touch the surface while holding a given token, the specific multi-touch spatial pattern associated with it is recognized using a pattern-matching algorithm that does not require any training or calibration. TouchTokens are fully passive. They can be fabricated using any non-conductive material, offering designers much flexibility in that respect. However, the proposed approach is currently limited to a set of simple shapes (square, rectangle, circle and triangle). In this article, we introduce two tools that allow interface designers to build and recognize TouchTokens featuring arbitrary shapes.

Our first contribution, *TouchTokenBuilder*, is a software application that assists interface designers in placing notches on arbitrarily-shaped vector contours for creating *conflict-free* token sets. The application features a simple direct-manipulation interface and outputs two files: a vector-graphics description of all tokens in the set, ready to be fabricated using, *e.g.*, a laser cutter; and a numerical description of the geometry of each token.

Our second contribution, *TouchTokenTracker*, is a software library that takes as input the numerical description produced by *TouchTokenBuilder*. While TouchTokens' original algorithm [19] only provided developers with the ID of the recognized token and the user's finger coordinates, the new *TouchTokenTracker* also enables tracking the tokens' full geometry (location, orientation and shape) throughout their manipulation on the multi-touch surface. In addition, tracking remains robust even when users lift a finger while manipulating tokens (leaving a minimum of two fingers in contact with the surface), as illustrated in Figure 1.

After reviewing related work, we describe *TouchTokenBuilder* and *TouchTokenTracker*. We then present some proof-of-concept token sets designed with *TouchTokenBuilder*, and report on experiments conducted to evaluate *TouchTokenTracker*'s recognition accuracy for these token sets. Finally, we discuss the limitations of our approach and directions for future work.

Caroline Appert & Emmanuel Pietriga & Éléonore Bartenlian & Rafael Morales González. Custom-made Tangible Interfaces with TouchTokens. In AVI '18: Proceedings of the International Working Conference on Advanced Visual Interfaces, 8 pages, ACM, may 2018.

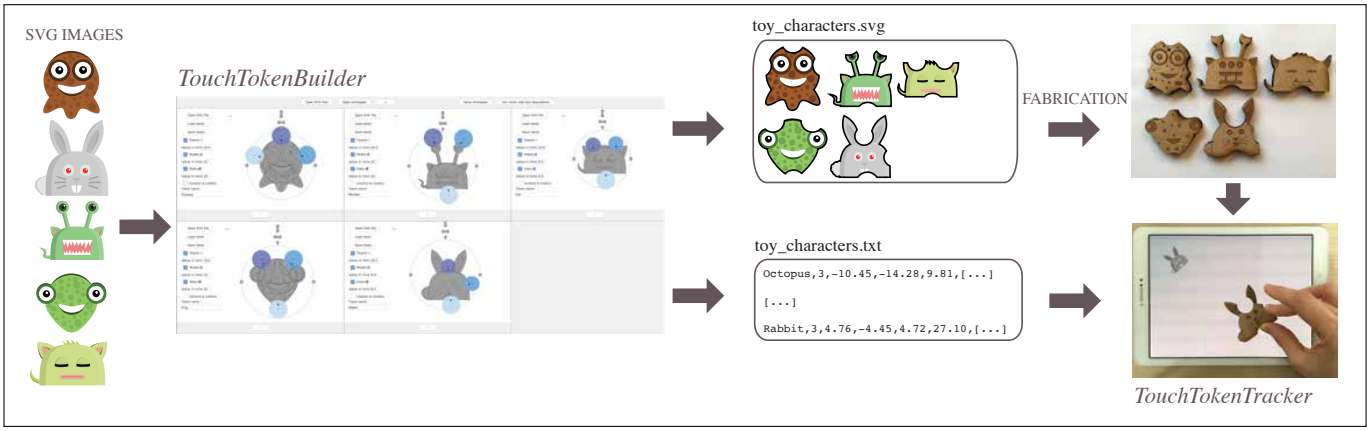

Figure 1: *TouchTokenBuilder* (left) assists users in placing grasping notches on arbitrarily-shaped tokens, warning them about spatial configurations that could generate recognition conflicts or that might be uncomfortable to manipulate. *TouchTokenBuilder* outputs both a vector and a numerical description of the tokens' geometry (middle). Those are used respectively to build the tokens (top-right), and to track them on any touchscreen using *TouchTokenTracker* (bottom-right).

2 RELATED WORK

Researchers have investigated different approaches to building tangible interfaces. We give an overview of these approaches and discuss their advantages and limitations, as well as their flexibility in terms of building custom-made interfaces.

Vision-based token tracking. Tangible interfaces that use a diffused illumination table (*e.g.*, [13, 14, 23]) or any other system that tracks the surface with one or several cameras (*e.g.*, [3, 5, 11]) rely on vision-based algorithms to recognize objects in video frames. While such vision-based approaches can be used to track any kind of tangible object, they require calibration as well as specific environmental conditions to avoid issues related to lighting and occlusion.

Active tokens. Active tokens (*e.g.*, [15, 24, 25, 30]) are small, autonomous units equipped with a processor and a screen that function independently from any specific interactive surface. Active tokens are programmable units, which can be customized for any type of application. Customizing their form factor can be achieved by constructing specific casings [15, 30], but the design space remains limited, as casings have to accommodate the incompressible active unit.

Magnetic tokens. Communication between the surface and the tokens can be achieved using magnetic fields (*e.g.*, [12, 17, 18]). Building a custom token means embedding a magnet and implementing an algorithm for recognizing its specific magnetic field. Gauss bricks [17] can be assembled together to create larger objects that can feature both deformable and rigid parts. While this approach enables very rich interactions, it still requires augmenting the surface with Hall-sensors and ensuring that the device's environment is free of any ferrous object that could interfere with the tangibles' magnetic field.

Capacitive tokens. Multi-touch capacitive screens can also be used to transform tokens into interactive elements. The approach consists of building tokens that create a conductive circuit between users' fingers and the capacitive surface through the tokens' *feet*, that are in contact with the surface (*e.g.*, [7, 16, 28]). As soon as the user touches the token, the feet become grounded and generate a drop in capacitance similar to a multi-touch pattern. Designing such tokens requires identifying unique token feet configurations and building a robust conductive circuit, which may be quite difficult [4].

TouchTokens. TouchTokens provide designers with a low-cost, flexible approach to the construction of tangibles using everyday materials. The original approach [19] was limited, however, because of the fact that the only input data available to the system consisted of the coordinates of the users' three finger contact points. Only one set of six basic tokens were available, and interface developers only had access to the recognized token's ID and associated finger contact points. In this paper, we extend TouchTokens so as to enable the design of tangible interfaces that feature arbitrarily-shaped tokens while preserving the simplicity of the original approach. We achieve this extension without resorting to any additional input technology: only the finger contact point coordinates are required to recognize and track tokens.

3 TOUCHTOKEN BUILDER

TouchTokens feature three notches that suggest to users how those tokens should be grasped so as to enable effective recognition of those tokens by the system. The recognition algorithm only needs one unique template per token, called *universal template*. This template consists of a series of three coordinates, that correspond to the expected finger contact point coordinates relative to the token's center. These simple templates have been demonstrated in [19] to be sufficient to achieve a recognition accuracy of ~98% with a set of six tokens featuring basic geometrical shapes (4-to-5cm large).

TouchTokens' approach is simple. But it requires designers to compute the coordinates of the templates' points (feeding the recognizer), and to specify the geometry of each token's contour with some vector-drawing tool (for fabricating the tokens), carving them accordingly to create the notches. This can be a tedious process.

This section introduces *TouchTokenBuilder*[1], an application that makes the token design process easier. Building a token now simply consists of importing an SVG image, and positioning the three notches on that contour by dragging three circles that represent the user's finger tips.

3.1 Designing arbitrarily-shaped tokens

Figure 1 illustrates the general approach that a designer can follow when creating a set of tokens, in this case for a game where toy characters (octopus, monster, cat, frog and rabbit) are controlled with tangible tokens. He first identifies a set of SVG images he wants to use for the different tokens. In our scenario, those simply get downloaded from the Web.[2] He then loads them in *TouchTokenBuilder*. For each SVG image, *TouchTokenBuilder* computes the outline of the entire geometry and creates a new cell in which it displays SVG elements (ignoring style attributes to avoid visual interference with *TouchTokenBuilder*'s graphical elements), as illustrated in Figure 1. SVG elements are turned into Java2D shapes with the help of the Batik toolkit.[3] The outline of the entire geometry is computed by making the union of all those shapes, taking into account groupings and affine transforms, and then marching along the contour of the resulting shape.

Each token outline can be manipulated using simple widgets to adjust its scale and orientation. As shown in Figure 2-(a), a ring surrounds the token, featuring two square handles to resize the token, and a circular handle to rotate it. Two arrows, positioned above, let users flip the token vertically or horizontally. A panel on the left-hand side of each token cell enables users to position the three finger notches on the outline. Fingers are represented using semi-transparent blue circles (thumb: light blue, middle: medium blue, index: dark blue). Each of these circles can be dragged and resized, and acts as a carving tool: when a circle intersects the token's outline, it actually subtracts the intersecting area from the token, computes the corresponding universal-template point (*i.e.*, the estimated finger contact point), and detects potential sources of conflicts between tokens, as detailed later.

TouchTokenBuilder adapts each token's display size depending on screen resolution, so that it matches its actual physical size when fabricated. Such resolution independence helps designers informally evaluate how comfortable a given token is to grasp, by putting their fingers on the corresponding circles on screen. The SVG vector description output by *TouchTokenBuilder* declares the document size and view box parameters so that the coordinates are correctly interpreted by the fabrication device that will be used to make the tokens such as, *e.g.*, a laser cutter.

Once satisfied with his set, the designer can export the corresponding vector and numerical descriptions (Figure 1-middle). *TouchTokenBuilder* turns what was a tedious process (relying on vector graphics editing and geometrical computations) into a sequence of simple, direct manipulations. It does not require users to manually draw or extract the token's contour. Most importantly, it enables users to very easily test alternative placements for the finger notches, as the computation of the carved contour is now fully automatic. Token design is achieved through a very simple interaction model, based exclusively on drag-and-drop, that avoids premature commitment [9], making the design process much more flexible. To further facilitate exploratory design by trial-and-error, *TouchTokenBuilder* also supports a per-object history of actions [1], enabling users to revert any graphical object such as, *e.g.*, a finger circle or a token manipulation handle, to one of its earlier positions.

3.2 Anatomical considerations

TouchToken assumes by design that users will be holding tokens using a three-finger grasp (*i.e.*, "tripod" grasp), which has been shown to be the typical grasp for simple objects whose diameter is ~4-7cm [8]. Based on this tripod-grasp assumption, *TouchTokenBuilder* provides users with some indications about the stability and comfort of each token grasp.

Stability. Research in experimental psychology has produced a model according to which, in a tripod grasp, the thumb acts in opposition to the other fingers to form a pinch [2]. *TouchTokenBuilder* estimates the forces applied by the different fingers as follows: i) the thumb applies a force towards the tripod's centroid, and ii) both the index and the middle apply a force towards the thumb. A grasp is likely to be ineffective at firmly maintaining the token during manipulation if one of the forces has a direction that is too similar to the tangent to the token contour at that point. As soon as the three fingers are placed along the token contour, *TouchTokenBuilder* displays arrows to represent the forces. Arrows' color depends on the angle they form with the token contour: from green (orthogonal) to red (parallel) (using a linear interpolation of H, and keeping S and V constant in the HSV color space). For example, Figure 2-(b) shows a token that is likely to be unstable during manipulation, as the index (dark blue) and middle finger (medium blue) might slip along the token contour.

Comfort. In *TouchTokenBuilder*'s interface, designers are free to move fingers anywhere along the token contour. Nothing prevents them from defining tripods that are very uncomfortable, or even impossible, to achieve. Fingers are not independent entities; each finger's range of motion heavily depends on the other fingers' position [10, 29]. In order to identify comfortable tripod configurations, we ran an experiment to estimate the range of comfortable positions for the index finger once the thumb and the middle finger are positioned on the surface. Figure 3 illustrates our experimental task. Participants hold a physical ruler between their thumb and middle fingers, and then slide their index on the surface to color the area that they deem comfortable. We chose this setup, where the index is mobile and the middle finger is static, because the degree of dependence of the middle finger is higher than that of the index [10]; meaning that moving the middle finger once the location of the index is set is more difficult than the opposite.

Six right-handed volunteers (3 female), aged 26 year-old on average (median: 26.5), participated in our experiment. The tablet was a Samsung GT-P7510 with a 217 × 137 mm display area and a resolution of 1280 × 800 pixels. Each participant performed 5 trials with 8 different rulers of varying lengths ($Length_{ruler} \in$ {3cm, 4cm, ..., 9cm, 10cm}) for a total of 40 trials, which were presented in a random order. We considered different ruler lengths as we had

[1] *TouchTokenBuilder* Java application and *TouchTokenTracker* TUIO and Android implementations are available at https://www.lri.fr/~appert/touchtokens.
[2] In this particular example: http://www.clipartlord.com/category/halloween-clip-art/monsters-clip-art/
[3] http://xmlgraphics.apache.org/batik/ (visited 2018-01-12)

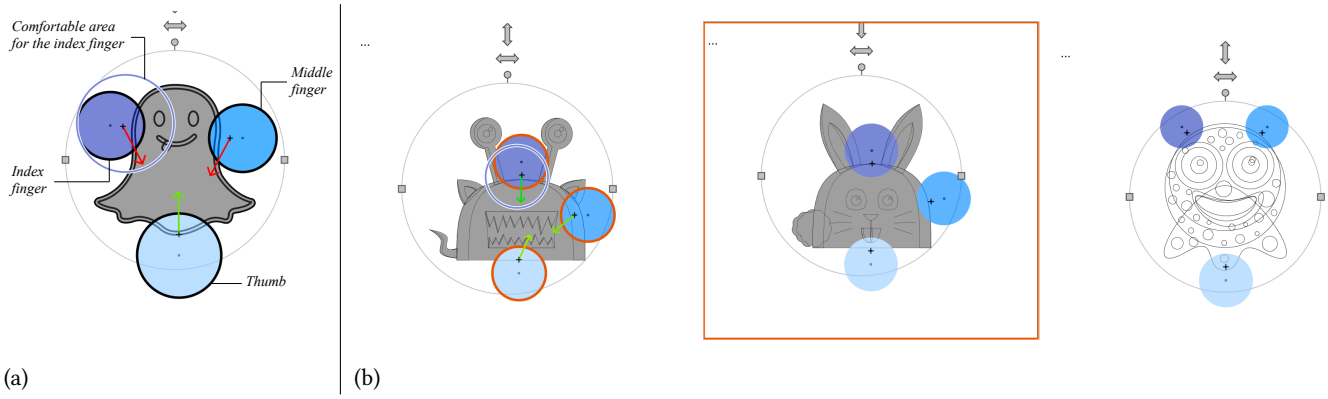

Figure 2: *TouchTokenBuilder* provides immediate visual feedback about both finger tripod comfort and potential recognition conflicts between tokens. *TouchTokenBuilder*'s interface distinguishes the token currently manipulated by the user (the active token) from the other tokens. (a) An active token, with arrows representing forces (two of which are colored red to indicate that the grasp might be difficult to maintain) and a blue hollow circle representing the comfortable area for the index finger (the grasp should be comfortable as long as the black cross corresponding to that finger remains inside this circle). (b) Token conflict: the active token (left) is currently in conflict with the token whose frame is colored red (middle one). Notch circle contours for the active token are also painted red to further emphasize this conflict, and suggest that the conflict can be resolved by moving one or more of those notches.

Figure 3: A trial to collect comfortable tripod grasps. (Left) Two circles indicate where to put the physical ruler (red circle: middle, blue circle: thumb). (Right) Participants slide their index to turn yellow the area within which this latter finger can be without making the hand posture become uncomfortable.

hypothesized that the distance between the thumb and the middle finger might have an impact on the range of motion of the index finger. As illustrated in Figure 3, at the beginning of each trial, two colored circles indicate where to put the ruler. Participants then color the comfortable area with their index. In case either the thumb or the middle finger leave their tolerance area, the participant is asked to restart the trial.

We wanted *TouchTokenBuilder* to be able to provide recommendations about where to put the notch for the index finger once the two other notches were positioned on the token. We thus use descriptive features of the comfortable area for the index finger (I_{area}, the polygon envelope for the polyline defined by sliding movements) that are relative to TM, *i.e.*, the segment defined by the thumb's location (T) and the middle finger's location (M). The list of features is as follows:

- $|TI|$: the distance between the thumb and the center I of I_{area}'s bounding box,
- I_{radius}: the radius of the largest circle inscribed in I_{area}'s bounding box,
- $\angle MTI$: the angle formed by points M, T and I at point T.

Collected data reveal a lot of variability across participants, with the following average values:

- $|TI|$: min average value = $3.9 \pm 1cm$, max = $6.3 \pm 1.8cm$;
- I_{radius}: min average value = $1.3 \pm 0.2cm$, max = $1.7 \pm 0.5cm$;
- $\angle MTI$: min average value = $51° \pm 10°$, max = $60° \pm 7°$.

We wanted to test if this variability came from differences in finger size across participants, as this would have enabled us to define the comfortable area as a function of finger size. To that end, we considered variable $Finger_{size}$, which is the average size over the three fingers involved in tripods,[4] and tested its effect on our measures. An anova test revealed that $Finger_{size}$ has a significant effect on $|TI|$ ($F_{5,234} = 20.3$, $p = 0.001$). However, further investigations showed that differences in $|TI|$ do not only come from differences in finger size, as we also observed a significant effect of $Finger_{size}$ on ratio $|TM|/|TI|$ ($F_{5,234} = 63.7$, $p = 0.001$).

This variability shows that the estimation of comfortable tripods should be made carefully. Ideally, an application that aims at providing recommendations should base them on individual hand measurements gathered through, *e.g.*, a short calibration phase with a setup similar to that of the experiment described above. However, not only would this add some overhead, but it would also require that tokens be designed by end-users. We adopted a less precise but more versatile approach. We aggregate data from our different participants to define an *average user*, and we use these data to display a circular area that gives a coarse indication of where the index should be put once the thumb and the middle finger are positioned (Figure 2). We insist on the fact that it gives only a coarse

[4]For each participant, we measured the distance between the bottom of the proximal phalanx and the top of the distal phalanx for the three fingers (thumb, middle, index).

indication, and encourage users to place their fingers on the screen and use their personal judgment to evaluate a tripod's comfort.

3.3 Recognition conflicts

As described above, *TouchTokenBuilder* makes it easy for designers to test different positions for the three notches that must be carved in each token. However, finding correct positions for notches is not solely a question of comfort and aesthetics (avoid altering the original shape too much). It also involves preventing recognition conflicts between tokens in the set. To this end, *TouchTokenBuilder* provides immediate visual feedback about conflicts. The contours of finger placeholders for the active token (the one currently being manipulated) change color (smoothly, in a range from green to red) when conflicts with other tokens are detected. So does the frame of the most-conflicted token. Figure 2-(b) illustrates a rather strong conflict (red contour color). Resolving such conflicts can be achieved by moving one or more notches along the contour, or adjusting the token's size, thereby causing the notches to move closer or farther away from the token's centroid. Visual warnings (red contours) disappear as soon as the conflict has been resolved.

Conflict detection is based on a heuristic derived from data collected in the second experiment reported in [19]. For all three notches of each trial, we computed the distance between the actual touch point and the template point $P_{template}$, located 5mm away along the normal at the notch's center. Figure 4 summarizes the results: in ~98% of all cases, this distance is less than 5mm for all three notches. Based on these observations, we define the tolerance area of a notch as a 5-mm radius circle around its corresponding template point. Two tokens are thus likely to cause confusion if they can accommodate the same multi-touch input within their respective tolerance areas.

TouchTokenBuilder relies on this notion of tolerance area to check for conflicts each time the contour of a token T in the pair is modified (the user has changed the token orientation or size, or a notch location or size). The algorithm for conflict checking works as follows. First, for each pair of tokens (T, T'), it aligns the template of T' with that of T, and computes the distance between the two other pairs of points ($dist_1$ and $dist_2$, in cm). If at least one of those distances is greater than 1cm, there is no conflict, and *TouchTokenBuilder* does not perform any further check. Otherwise, the probability for T' to conflict with T is computed as 1 - max($dist_1$, $dist_2$). *TouchTokenBuilder* looks for the token that has the highest conflicting score s with T, and highlights both the frame of this token and the contour of finger placeholders in T using the green-to-red color range mentioned earlier.

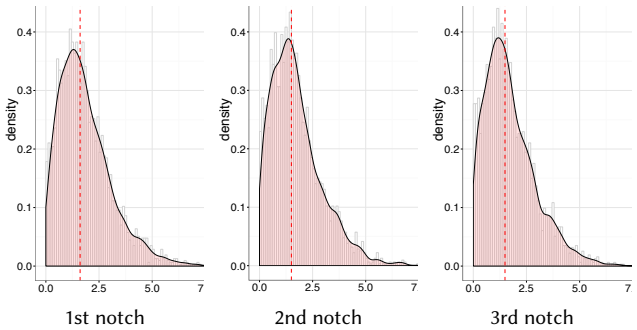

Figure 4: Distance (mm) between $P_{template}$ and P_{actual} (template and actual touch points) for all 3 notches. Red dashed lines show median values.

4 TOUCHTOKEN TRACKER

TouchTokenTracker allows developers who make use of arbitrarily-shaped tokens in their application to track the full geometry of those tokens. Distributed as a library, it enables the development of applications that need to display contextual information around or below the token. Examples include information filtering using tangibles as see-through tools [6], and games (Figure 6-(c) demonstrates launching missiles from a tangible spaceship). *TouchTokenTracker*'s recognition algorithm relies on the three points provided in each token template, as the original TouchToken recognizer [19] did. It considers two additional pieces of information, provided in the new templates output by *TouchTokenBuilder*: the token's center coordinates, and a description of its contour as a polyline. These are used to estimate the location and orientation of the token, which are then exposed through a simple API.

To recognize tokens, *TouchTokenTracker* identifies the best alignment between the points of each candidate token's template and the actual touch points. Aligning template points with touch points requires translating and rotating template points so as to (1) make the centroids of the touch and template points coincide, and (2) align this centroid with the first pair of matched touch and template points. *TouchTokenTracker* stores the pairing between a touch point and its corresponding template point. It also stores the initial locations of the touch points and the token's initial orientation, which is the rotation angle used to align the first pair of points with the centroid. Using this information, it can estimate the current orientation and location when users move and rotate the token. In case the user lifts a finger off the surface to adopt a 2-finger pinch grasp that facilitates some manipulations, as in Figure 6, *TouchTokenTracker* computes a third artificial touch point, assuming that the relative placement between touch points and between the template points are consistent. Keeping track of the three notches' locations can be useful to implement some interactions like the missiles launched by the spaceship in Figure 6-(c).

Events and information regarding a token geometry are made available to developers through three simple callbacks: `tokenDown`, `tokenMoved` and `tokenUp`, and methods: `getTouchPoints`, `getNotchPoints`, `getContourShape`, `getInitialOrientation`, `getRelativeOrientation` and `getTokenCenter`.

5 EXPERIMENT 1: DESIGNING TOKENS

We conducted an experiment to observe whether users are able to design conflict-free and comfortable token sets without significant training. The experiment consisted of two sessions, held on two consecutive days. On day 1, participants had to design a token set with *TouchTokenBuilder* on a large horizontal screen, starting from one of the three picture sets shown in Figure 5. We then fabricated that token set using a laser cutter. On day 2, participants had to perform a series of docking tasks with their tokens on a tablet device. The goal was to evaluate recognition accuracy for this particular token

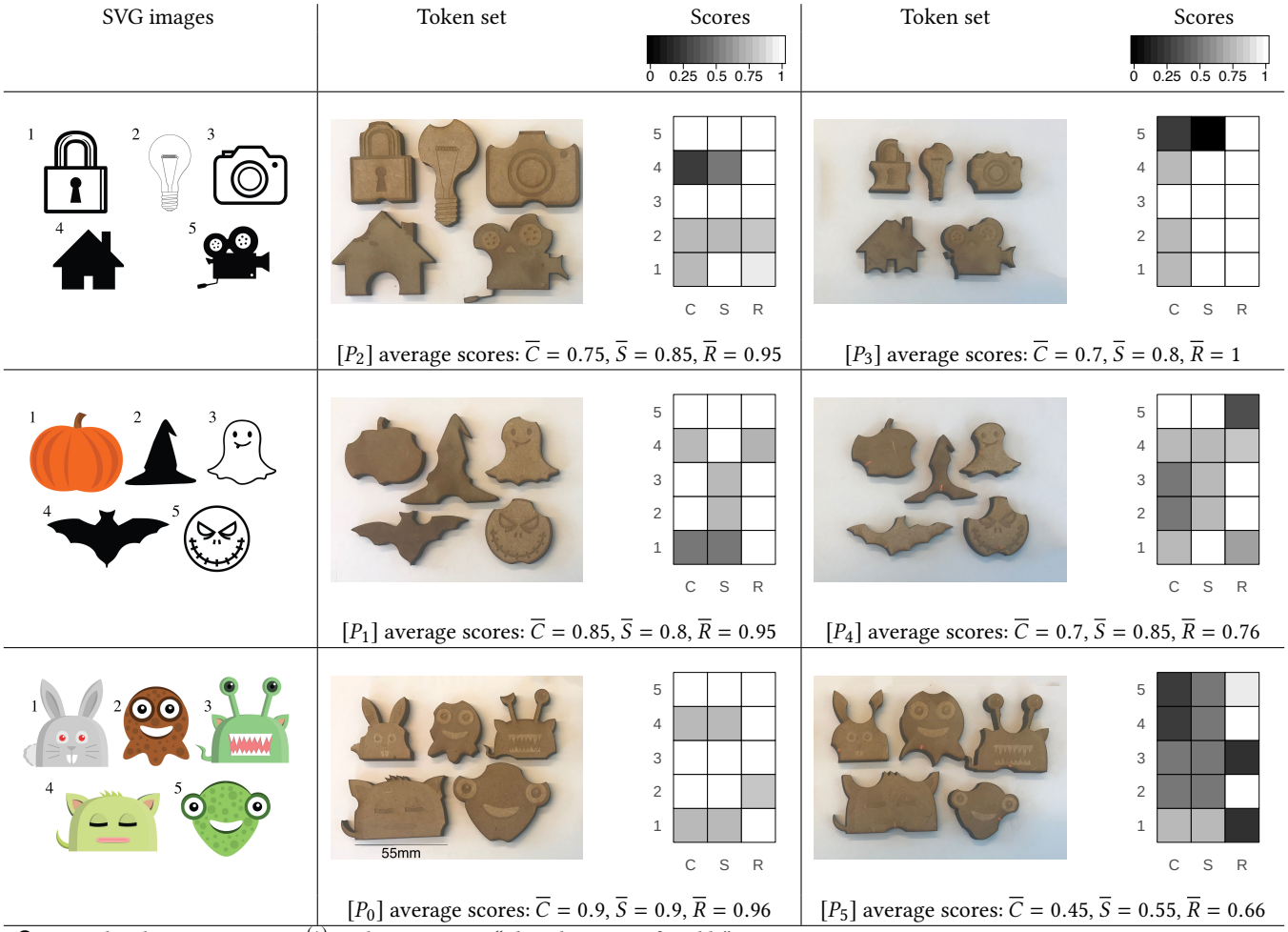

Figure 5: The six token sets that participants designed during the experiment, along with their scores.

set, and to gather feedback about how comfortable individual tokens are to grasp. The docking task (Figure 7-(a)) and experimental procedure is identical to that of Experiment 2, reported in the next section. For this first experiment, which focuses on token design, participants also had to give a comfort score C, and a stability score S, by rating the two statements reported in Figure 5 on 5-point Likert scales. The last score, R, is the recognition accuracy that we measured on day 2, which involved five docking tasks per token.

Participants. Six volunteers (three male), all right-handed, aged 25 to 41 year-old (median 29.5), participated in our study. Four of them had already interacted with TouchTokens in the context of the study reported in [19]. The other two (P_4 and P_5) had only watched the companion video of [19].

Apparatus. On day 1, *TouchTokenBuilder* was running on a 3M Multi-Touch Display C3266P6, featuring a 698.4 × 392.85 mm display area and a resolution of 1920 × 1080 pixels. The display was placed flat on a desk, in landscape orientation. On day 2, the experiment software was running on a Samsung SM-T810 Galaxy Tab S2, featuring a 237.3 × 169 mm display area and a resolution of 2048 × 1536 pixels. It was also placed flat on a desk.

Task. At the beginning of the design session on day 1, the operator gives a 5-minute demonstration of *TouchTokenBuilder* on a token set that will not be used in the remainder of the experiment. The operator explains the meaning of the different visual indicators, and insists on the fact that those are just estimations, inviting participants to place their fingers on-screen to cross-check these estimations with their personal appreciation. Then, participants are given 20 minutes to place notches on the five tokens in the set. They can stop at any moment before this time has elapsed, if they are satisfied. We chose to test sets of five tokens, as our personal experience revealed that having to deal with a larger number of arbitrarily-shaped tokens makes the task especially challenging.

Results. Participants took from 11 to 20 minutes (median: 17) to complete the task. Figure 5 provides an exhaustive report of our results, per participant. The average scores were rather good, with i) a recognition accuracy (R) of 88% (min: 66%, max: 100%); ii) a comfort score (C) of 0.73 (min:0.45, max:0.9), indicating good agreement with statement *"The token is comfortable"*; and iii) a stability score (S) of 0.8 (min:0.45, max:0.9), indicating good agreement with statement *"The token is stable during manipulation"*. However, there is a lot of variability across participants. In particular, P_4 and P_5, who had never manipulated TouchTokens before, had more difficulty designing comfortable and conflict-free token sets. As opposed to other participants, they both mentioned that they had some trouble imagining what the physical tokens would eventually look and feel like. One of them said that having an up-to-date view of the resulting tokens, as stored in the SVG export (as opposed to only seeing their silhouettes in *TouchTokenBuilder*'s interface) would have helped. Regarding anatomy, P_5, who has especially large hands, asked for the possibility to calibrate the application in order to get comfort indications that better fit different types of hands. Both P_4 and P_5 also mentioned that they would have liked to involve the ring finger, either as a fourth finger in the grasp, or as a replacement for the index finger. The other participants, who had already manipulated TouchTokens, never brought up such issues. However, our results support the fact that 5 is close to the maximum number of arbitrarily-shaped tokens that can be managed, as almost all sets feature a token that is either less-accurately recognized or less comfortable than the others. Participants also raised an issue with the colors used for the different finger notches, that were found to be too similar to each other. Finally, two participants suggested that, when *TouchTokenBuilder* runs on a multi-touch surface, it could allow users to simply put their fingers on screen to provide a first grasp estimation, that could then be adjusted to resolve conflicts and make them comfortable.

6 EXPERIMENT 2: TRACKING TOKENS

This section presents an evaluation of *TouchTokenTracker*'s accuracy on three token sets that we developed for the proof-of-concept applications shown in Figure 6 and in the companion video.[5] The first proof-of-concept example is about controlling a virtual toy character using its tangible counterpart. The second example is a simplified house automation control system. Users can switch the light on/off, get information about energy consumption, turn on video-surveillance, play music, and lock the house. The last example is a bi-manual game. Users manipulate one of the warships with their dominant hand, and select a weapon with their other hand.

We tested whether *TouchTokenTracker* could accurately identify the tokens' location and orientation. We ran one experiment per token set {*Toys*, *Home*, *Space*}. Each experiment had two factors: Token and Orientation. *Toys* and *Home* each featured five tokens, while *Space* featured four tokens. Orientation could take five different values: $\{-\pi/3, -\pi/6, 0, +\pi/6, +\pi/3\}$. Values outside $\{-\pi/3, +\pi/3\}$ were not considered, as they would have been beyond the limits of users' range of motion.

[5]The companion video is available at https://www.lri.fr/~appert/touchtokens.

Participants. Nine volunteers (one female), aged 23 to 33 year-old (average 26.5, median 26), participated in our study. Each of them performed the three experiments (one per token set) in a row.

Apparatus. The experiment ran on a tablet (Samsung SM-T810 Galaxy Tab S2) featuring a 237.3 × 169 mm display area and a resolution of 2048 × 1536 pixels. Participants were standing up, holding the tablet during the whole experiment.

Task and procedure. At the start of each trial, a token silhouette was displayed at the center of the screen, with a specific orientation (Figure 7-(a)). Participants were asked to dock the corresponding physical token inside the silhouette, and wait for the background to turn blue before lifting the token off the surface, and proceed to the next trial. Participants had to hold the token still for 1 second. *TouchTokenTracker*'s algorithm was then executed. The system logged the ID of the recognized token, its estimated location, and orientation.

We counterbalanced token-set presentation order using a Latin-square, assigning three participants to each of the three orders. For each token set, participants ran 5 trials per Token, testing the 5 Orientation values. The experiment was approximately 10-minute long. It started with 5 practice trials (randomized Token × Orientation conditions), followed by 25 measure trials (20 for *Space*) presented in random order.

Results. We considered the following three measures to capture *TouchTokenTracker*'s accuracy: *RecognitionError*, a binary value indicating whether the token is accurately recognized or not; *OrientationError*, a continuous measure of the absolute difference (in radians) between the silhouette's orientation and the physical token's orientation, as estimated by *TouchTokenTracker*; and *DistanceError*, a continuous measure of the distance (in millimeters) between the silhouette's center and the physical token's center, again as estimated by *TouchTokenTracker*.

Figure 6: The three proof-of-concept token sets: (a) toy characters, (b) home automation, (c) spaceship game.

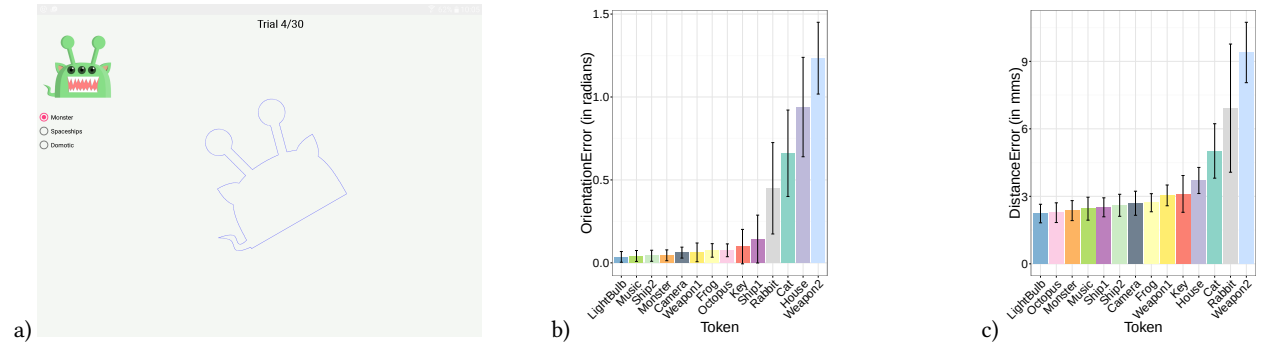

Figure 7: A trial in our experiment: (a) the participant has to dock the corresponding physical token inside the displayed silhouette. (b) *OrientationError* and (c) *DistanceError* per TOKEN. Error bars represent the 95% confidence interval.

We observed an overall recognition accuracy of 98%. The recognizer failed to identify the correct token in only 12 of the 630 trials: 6 times with the CAT, 3 times with the CAMERA, and once with the KEY, the RABBIT and WEAPON1. A Friedman rank sum test revealed that the difference between TOKEN conditions regarding recognition accuracy is actually significant ($\tilde{\chi}^2(13) = 38$, $p < 0.001$). We attribute this difference to the fact that the CAT token requires users to spread the index and middle fingers a bit too much. Participants might have placed their index and middle fingers closer together, so as to make their grasp more comfortable, thus not exactly coinciding with the notches.

Figure 7 summarizes the tracker's performance results regarding the evaluation of token position and orientation, a piece of information that the original recognizer [19] was unable to provide (as it was just providing the token's ID and the location of the fingers that were in contact with the surface). For 10 of the 14 tokens, *OrientationError* is less than 0.15 ($\frac{\pi}{20}$) and *DistanceError* is less than 3.1mm. However, *TouchTokenTracker*'s estimations are much less accurate for the other 4 tokens: HOUSE, CAT, RABBIT and WEAPON2. This result is not really surprising. These four tokens feature at least two template points that are symmetric relative to the axis defined by the third point and the centroid, which implies that there is more than one solution for the recognizer's best-alignment algorithm. While this does not affect the recognition of the token's ID, Figure 8 illustrates how two different orientations can match the same template points. As the token's center location is derived from the token's orientation, it is not surprising that *DistanceError* is also larger for those four tokens than it is for the other ten, whose orientation was properly estimated by *TouchTokenTracker*. We computed a linear regression to predict *DistanceError* from *OrientationError*. We found a significant relation ($F(1,616)$, $p < 0.001$) with r^2=0.53. We acknowledge this limitation of our approach, which is due to the fact that it relies on passive tokens and thus on what can be inferred from the three finger contact points only. However, this limitation can be alleviated by eliminating a range of unlikely orientations that fall out of users' range of motion, possibly warning users if the manipulated token still features an axis of symmetry.

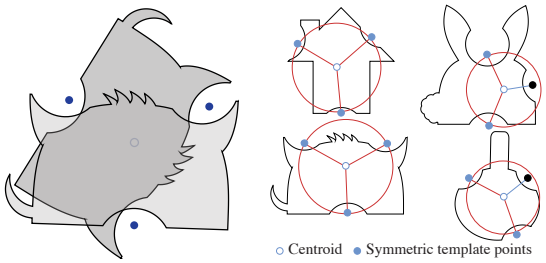

Figure 8: Token orientation: (left) an example of ambiguity; (right) error-prone tokens: HOUSE, RABBIT, CAT, WEAPON2.

7 DISCUSSION AND FUTURE WORK

Taken together, *TouchTokenBuilder* and *TouchTokenTracker* enable designers to build low-cost tangible interfaces using TouchToken while addressing several limitations of the original approach. This article shows that these tools enable the design and tracking of sets that feature up to five arbitrarily-shaped tokens. Each of the two tools, however, still has its own limitations, several of which can be addressed in future work.

TouchTokenTracker provides an estimate of each token's location and orientation, but these can be wrong in some cases. As shown earlier, we can eliminate high-amplitude errors, but there will still remain some uncertainty. This latter limitation results from the trade-off between accuracy and ease-of-implementation: relying on fully passive tokens makes building tangible interfaces easy, but requires the system to infer a lot from very few input data, which are limited in our case to the fingers' contact points.

TouchTokenBuilder lets users freely position notches on the tokens, and warns them about potential conflicts, and about uncomfortable grasps. However, our results show that enabling users to calibrate comfort recommendations based on their own hands would be a nice addition to *TouchTokenBuilder*. We also observed variations in the ease of use of *TouchTokenBuilder* depending on whether designers had already used TouchTokens or not. Because TouchTokens are very low-cost, we encourage designers to build the set of basic TouchTokens described in [19] in order to get a first experience with the approach. The feedback that we collected could also help improve *TouchTokenBuilder* with, *e.g.*, the ability to get a view of the SVG export at any time during the design, or the use of multi-touch input (when available) to position notches. Finally, we plan to extend *TouchTokenBuilder* in order to enable designers to easily make flexible versions of their tokens [20].

REFERENCES

[1] Caroline Appert, Olivier Chapuis, Emmanuel Pietriga, and María-Jesús Lobo. 2015. Reciprocal Drag-and-Drop. *ACM Trans. Comput.-Hum. Interact.* 22, 6, Article 29 (Sept. 2015), 36 pages. https://doi.org/10.1145/2785670

[2] Michael A. Arbib. 1990. Programs, schemas, and neural networks for control of hand movements: Beyond the RS framework. *Attention and performance 13: Motor representation and control.* (1990), 111–138.

[3] Daniel Avrahami, Jacob O. Wobbrock, and Shahram Izadi. 2011. Portico: Tangible Interaction on and Around a Tablet. In *Proc. UIST '11*. ACM, 347–356. https://doi.org/10.1145/2047196.2047241

[4] Rachel Blagojevic and Beryl Plimmer. 2013. CapTUI: Geometric Drawing with Tangibles on a Capacitive Multi-touch Display. In *Proc. INTERACT '13*. Springer, 511–528.

[5] David Bouchard and Steve Daniels. 2015. Tiles That Talk: Tangible Templates for Networked Objects. In *Proc. TEI '15*. ACM, 197–200. https://doi.org/10.1145/2677199.2680607

[6] Wolfgang Büschel, Ulrike Kister, Mathias Frisch, and Raimund Dachselt. 2014. T4 - Transparent and Translucent Tangibles on Tabletops. In *Proc. AVI '14*. ACM, 81–88. https://doi.org/10.1145/2598153.2598179

[7] Liwei Chan, Stefanie Müller, Anne Roudaut, and Patrick Baudisch. 2012. CapStones and ZebraWidgets: Sensing Stacks of Building Blocks, Dials and Sliders on Capacitive Touch Screens. In *Proc. CHI '12*. ACM, 2189–2192. https://doi.org/10.1145/2207676.2208371

[8] Maurizio Gentilucci, Luana Caselli, and Claudio Secchi. 2003. Finger control in the tripod grasp. *Experimental Brain Research* 149, 3 (01 Apr 2003), 351–360. https://doi.org/10.1007/s00221-002-1359-3

[9] Thomas R. G. Green and Marian Petre. 1996. Usability analysis of visual programming environments: a "cognitive dimensions" framework. *JVLC* 7, 2 (1996), 131–174. https://doi.org/10.1006/jvlc.1996.0009

[10] C. Hager-Ross and M.H. Schieber. 2000. Quantifying the independence of human finger movements: comparisons of digits, hands, and movement frequencies. *Journal of Neuroscience* 20, 22 (2000), 8542.

[11] Michael S. Horn and Robert J. K. Jacob. 2007. Designing Tangible Programming Languages for Classroom Use. In *Proc. TEI '07*. ACM, 159–162. https://doi.org/10.1145/1226969.1227003

[12] Sungjae Hwang, Myungwook Ahn, and Kwang-yun Wohn. 2013. MagGetz: Customizable Passive Tangible Controllers on and Around Conventional Mobile Devices. In *Proc. UIST '13*. ACM, 411–416. https://doi.org/10.1145/2501988.2501991

[13] Hans-Christian Jetter, Jens Gerken, Michael Zöllner, Harald Reiterer, and Natasa Milic-Frayling. 2011. Materializing the Query with Facet-streams: A Hybrid Surface for Collaborative Search on Tabletops. In *Proc. CHI '11*. ACM, 3013–3022. https://doi.org/10.1145/1978942.1979390

[14] Sergi Jordà, Günter Geiger, Marcos Alonso, and Martin Kaltenbrunner. 2007. The reacTable: Exploring the Synergy Between Live Music Performance and Tabletop Tangible Interfaces. In *Proc. TEI '07*. ACM, 139–146. https://doi.org/10.1145/1226969.1226998

[15] Stefanie Klum, Petra Isenberg, Ricardo Langner, Jean-Daniel Fekete, and Raimund Dachselt. 2012. Stackables: Combining Tangibles for Faceted Browsing. In *Proc. AVI '12*. ACM, 241–248. https://doi.org/10.1145/2254556.2254600

[16] Sven Kratz, Tilo Westermann, Michael Rohs, and Georg Essl. 2011. CapWidgets: Tangible Widgets Versus Multi-touch Controls on Mobile Devices. In *CHI EA '11*. ACM, 1351–1356. https://doi.org/10.1145/1979742.1979773

[17] Rong-Hao Liang, Liwei Chan, Hung-Yu Tseng, Han-Chih Kuo, Da-Yuan Huang, De-Nian Yang, and Bing-Yu Chen. 2014. GaussBricks: Magnetic Building Blocks for Constructive Tangible Interactions on Portable Displays. In *Proc. CHI '14 (CHI '14)*. ACM, New York, NY, USA, 3153–3162. https://doi.org/10.1145/2556288.2557105

[18] Rong-Hao Liang, Kai-Yin Cheng, Liwei Chan, Chuan-Xhyuan Peng, Mike Y. Chen, Rung-Huei Liang, De-Nian Yang, and Bing-Yu Chen. 2013. GaussBits: Magnetic Tangible Bits for Portable and Occlusion-free Near-surface Interactions. In *CHI EA '13*. ACM, 2837–2838. https://doi.org/10.1145/2468356.2479537

[19] Rafael Morales González, Caroline Appert, Gilles Bailly, and Emmanuel Pietriga. 2016. TouchTokens: Guiding Touch Patterns with Passive Tokens. In *Proc. CHI '16*. ACM, New York, NY, USA, 4189–4202. https://doi.org/10.1145/2858036.2858041

[20] Rafael Morales González, Caroline Appert, Gilles Bailly, and Emmanuel Pietriga. 2017. Passive Yet Expressive TouchTokens. In *Proc. CHI '17 (CHI '17)*. ACM, New York, NY, USA, 3741–3745. https://doi.org/10.1145/3025453.3025894

[21] Mikko Pyykkönen, Jukka Riekki, Marko Jurmu, and Iván Sanchéz Milara. 2013. Activity Pad: Teaching Tool Combining Tangible Interaction and Affordance of Paper. In *Proc. ITS '13*. ACM, 135–144. https://doi.org/10.1145/2512349.2512810

[22] Jinsil Hwaryoung Seo, Janelle Arita, Sharon Chu, Francis Quek, and Stephen Aldriedge. 2015. Material Significance of Tangibles for Young Children. In *Proc. TEI '15*. ACM, 53–56. https://doi.org/10.1145/2677199.2680583

[23] Yang Ting Shen and Ali Mazalek. 2010. PuzzleTale: A Tangible Puzzle Game for Interactive Storytelling. *Comput. Entertain.* 8, 2, Article 11 (Dec. 2010), 15 pages. https://doi.org/10.1145/1899687.1899693

[24] Brygg Ullmer, Hiroshi Ishii, and Robert J. K. Jacob. 2005. Token+Constraint Systems for Tangible Interaction with Digital Information. *ACM Trans. Comput.-Hum. Interact.* 12, 1 (2005), 81–118. https://doi.org/10.1145/1057237.1057242

[25] Consuelo Valdes, Diana Eastman, Casey Grote, Shantanu Thatte, Orit Shaer, Ali Mazalek, Brygg Ullmer, and Miriam K. Konkel. 2014. Exploring the Design Space of Gestural Interaction with Active Tokens Through User-defined Gestures. In *Proc. CHI '14*. ACM, 4107–4116. https://doi.org/10.1145/2556288.2557373

[26] Simon Voelker, Christian Cherek, Jan Thar, Thorsten Karrer, Christian Thoresen, Kjell Ivar Øvergård, and Jan Borchers. 2015. PERCs: Persistently Trackable Tangibles on Capacitive Multi-Touch Displays. In *Proc. UIST '15*. ACM, 351–356. https://doi.org/10.1145/2807442.2807466

[27] Simon Voelker, Kosuke Nakajima, Christian Thoresen, Yuichi Itoh, Kjell Ivar Øvergård, and Jan Borchers. 2013. PUCs: Detecting Transparent, Passive Untouched Capacitive Widgets on Unmodified Multi-touch Displays. In *Proc. ITS '13*. ACM, 101–104. https://doi.org/10.1145/2512349.2512791

[28] Neng-Hao Yu, Sung-Sheng Tsai, I-Chun Hsiao, Dian-Je Tsai, Meng-Han Lee, Mike Y. Chen, and Yi-Ping Hung. 2011. Clip-on Gadgets: Expanding Multi-touch Interaction Area with Unpowered Tactile Controls. In *Proc. UIST '11*. ACM, 367–372. https://doi.org/10.1145/2047196.2047243

[29] Vladimir M Zatsiorsky, Zong-Ming Li, and Mark L Latash. 2000. Enslaving effects in multi-finger force production. *Experimental Brain Research* 131, 2 (2000), 187–195.

[30] Jamie Zigelbaum, Michael S. Horn, Orit Shaer, and Robert J. K. Jacob. 2007. The Tangible Video Editor: Collaborative Video Editing with Active Tokens. In *Proc. TEI '07*. ACM, 43–46. https://doi.org/10.1145/1226969.1226978