Gesture Morpher: Video-Based Retargeting of Multi-Touch Interactions

Ramik Sadana^{* 1,2}
ramik@gatech.edu

Yang Li¹
yangli@acm.org

¹Google Inc., Mountain View, California

²Georgia Tech, Atlanta, Georgia

Figure 1. The Gesture Morpher interface. 1) Menu Bar allows users to record the screen of a connected Android device or import a local video file. 2) Blocks List presents blocks from the video where transformations are identified. 3) Video View displays the frames of the selected segment. 4) Timeline highlights the selected segments and lets users modify their boundaries. 5) Interaction List presents relevant interactions that users can test on a device.

ABSTRACT

We present Gesture Morpher, a tool for prototyping and testing multi-touch interactions based on video recordings of target application behaviors, e.g., a sequence of screenshots recorded by a screen capture tool. Gesture Morpher extracts continuous behaviors from video recordings, such as transformations of UI content, and suggests a set of multi-touch interactions that are suitable for achieving these behaviors. Designers can easily test different interactions on a touch device with visual response that is automatically synthesized from the video recording, all without any programming. We discuss a range of multi-touch interaction scenarios Gesture Morpher supports, our method for extracting continuous interaction behaviors from video recordings, and techniques for associating touch-input with the output effect extracted from the videos.

^{*}This work was done while the author was an intern with Google Research.

ACM Classification Keywords

H.5.2. Input Devices and Strategies, Prototyping.

Author Keywords

Multi-touch; video analysis; rapid prototyping; simulation

INTRODUCTION

The widespread reach and adoption of modern touch-based interfaces has fundamentally altered our interaction with the devices around us. Increasingly, traditional point-and-click-based WIMP interfaces are being transformed into touch-based interfaces (e.g., [12]). However, in spite of the presence of vocabularies and best practices for using multi-touch interactions [14], identifying appropriate interactions and prototyping them within applications remain challenging and time consuming.

We speculate that this challenge is a result of several fundamental issues that designers face. Developing interactions requires designers to have a minimum set of programming skills (e.g., knowing exact syntax and semantics of the language and platform they are using). Although recent work (e.g., [10]) eased the task of creating gesture input, it still requires a fair amount of programming effort to attach these behaviors to interface elements. Further, even with necessary programming skills, it is considerably tedious to iterate on multiple interaction alternatives that may be suitable for a behavior. Finally, there is a lack of mechanisms for making a

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

MobileHCI '16, September 06 - 09, 2016, Florence, Italy
ACM 978-1-4503-4408-1/16/09...\$15.00

DOI: <http://dx.doi.org/10.1145/2935334.2935391>

gesture vocabulary and best practices available to designers when designing novel interfaces.

A different approach to addressing the challenge of prototyping multi-touch interactions is to decrease the level of formal input required (e.g., syntax and semantics of the programming language). Instead of requiring designers to program various interaction alternatives, we could require designers to simply specify their intended behaviors (e.g., Zoom, Rotate) to the system. The system could then automatically find appropriate interactions for these behaviors and attach them to the elements of the interface. Such a system would allow a designer to quickly test the intended interactions within the context of an interface without writing any code. For example, if the user provides an input of the different states of a rotating image, the system would infer the rotation behavior, and automatically attach a Two-Finger-Rotate gesture to the image.

To demonstrate this technique, we present Gesture Morpher—a tool for prototyping multi-touch interactions (see Figure 1). Our tool uses video as a medium for specifying behaviors. The video contains the response of an interface to the interactions the designer intends to attach. Such a video can be generated by a screen-capture tool or any video authoring tool (e.g., Adobe After Effects). Our tool extracts the behaviors from the video, and recommends a collection of suitable interactions for each identified behavior. The recommendation is based on a gesture vocabulary we developed that maps application behaviors to appropriate gesture alternatives by matching their degrees of freedom. Users can select and test any suggested interaction directly on a connected touchscreen device. To provide in-context feedback, the visual response to the interaction is automatically synthesized from the video recording and is synchronized with the actual finger movement.

To scope our work, we focus on continuous interaction behaviors, such as geometric transformation of UI content, which are the primary class of behaviors where multi-touch interactions are used. This paper offers two key contributions:

- A set of methods for understanding a user interface and extracting continuous interaction behaviors, such as geometric transformation, from video recordings, and for associating multi-touch interaction input with the output effect captured in video frames.
- Integrated tool support for designers to easily test multi-touch interactions for a behavior, based on its video recording, without any programming. The tool “morphs” the video recording to synthesize the output effect in response to actual interaction input.

RELATED WORK

Specifying Interaction Behaviors

Extensive work has been conducted to reduce the effort of touch-based interactions development. Kin et al. [8] used regular expressions for developers to specify multi-touch interactions, alleviating the effort in dealing with programming details. Li et al.’s work [10] allowed developers to incorporate multi-touch interactions, both existing and novel, by directly demonstrating them on a target device. While these tools

simplify interaction construction for developers, substantial effort is still needed for linking these interactions with actions. Our focus with Gesture Morpher is to minimize this programming effort. Although our approach does not integrate directly within the systems as previous work does, it allows developers to rapidly test a variety of predefined touch-based interactions on an actual device with the desired UI feedback. Testing an interaction in the context of its use helps in a holistic assessment of its usability.

Reverse Engineering User Interfaces

The approach of analyzing screen captures (pixels) for understanding UI structures and behaviors has been previously used in a variety of situations. Yeh et al. [16] performed template matching to identify UI widgets in a screenshot for search and automation. Dixon and Fogarty [3] achieved precise real-time identification of UI elements for enhancing an existing interface with alternative input methods. Both these systems employ predefined models of UI components to identify target elements on the view. However, constructing such models for components typically provided in modern mobile UIs remains a challenge since these interfaces tend to follow *flat-design* principles. These principles specifically reject the use of borders and textures, which are needed to differentiate.

An approach to analyze interfaces without needing predefined templates was presented by Banovic et al. [1] in the design of their Waken system. Waken analyzes video tutorials and extracts dynamic behavior by creating templates to identify the cursors, icons, menus, and tooltips on the fly. We employ a similar approach in Gesture Morpher by looking at a sequence of screen captures without requiring any prior knowledge of the interface. The difference in the two approaches, however, is that the goal in Waken is to characterize the discrete interactions with widgets, while with Gesture Morpher our aim is to identify direct, continuous manipulation of UI content that multi-touch gestures are typically employed for.

For analysis of continuous manipulation within videos, typical techniques are based on feature tracking or optical flow. The most relevant prior work in perhaps Dragicevic et al.’s DimP [4], which extracts the motion path of an object in a real-world video, e.g., a moving soccer ball. A user can playback the video by directly dragging the object in the video along the motion path. Similar to DimP, we perform motion analysis on video and allow the designer to “playback” the video based on how the motion is aligned with the interaction input on a touchscreen device. However, our approach differs in several ways. Gesture Morpher processes the video recordings of UI changes that have far fewer visual features than the real-world video that DimP uses, and the lack of visual features raises challenges for motion tracking. In addition, DimP tracks a single type of motion and maps it to a single type of input movement (dragging). In contrast, we identify a range of motion types and map them to a variety of multi-touch interactions.

Touch-based Interaction

Touch interaction design has been studied extensively in previous work. Researchers have developed a vast body of knowledge exploring use of gestures for all category of operations

across contexts and use cases [11, 15]. In addition to touch-based interactions that are designed by experts, researchers have also presented user-elicited gestures for a large category of operations on touch-based devices [14]. However, there is a lack of mechanisms whereby designers can readily and systematically adapt these results and best practices within their actual design process. We intend to help designers realize these conceptual findings by automatically suggesting touch-based interactions appropriate for their tasks.

THE DESIGN OF GESTURE MORPHER

Although the set of possible interactions is fairly large, there is often only a limited set of interactions that are appropriate for a given behavior. A variety of properties of an interaction can determine its appropriateness for a task, such as familiarity or affordance. One such property that particularly motivates our work is the degree of compatibility, which measures the similarity between the gestural input and the response of the interface [2]. Employing interactions with a high degree of compatibility between input and output increases the directness of an interface [7]. The interactions that we commonly use today, such as Tap and Swipe, observe this principle.

Based on this principle of directness, Gesture Morpher automatically suggests interactions that are compatible with a specific type of UI response or behavior as demonstrated in a video. The video demonstrating these UI changes can be a sequence of frames generated from a UI mockup in tools such as Framer, Invision or Adobe After Effects, or manifested by the designer manipulating an existing user interface.

To help illustrate the use scenario of Gesture Morpher, assume Joan is designing an Android interface for map navigation. She mocks up the interface in Adobe Illustrator. As soon as she finishes a screen, she imports the Illustrator file in Adobe After Effects to animate the layers. This generates a set of video clips that show both the look-and-feel and the dynamic behaviors of the UI. Joan then wants to design multi-touch interaction for the UI mockup. To do so, she loads the video into Gesture Morpher, which automatically analyzes the dynamic behaviors demonstrated in the video and displays those transformations on the left panel of the Gesture Morpher interface as a list (see Figure 1-2). She can select any of the transformations, which highlights the corresponding segments of frames on a timeline at the bottom (see Figure 1-4). She can view the frames by playing the video, and modify the ends of a segment to include or exclude frames for the selected transformation. Selecting a transformation also reveals the associated interactions options that the system suggests.

Joan can select one of these interactions and test it in the context of the target behaviors, directly on a connected touch-screen device, such as a smartphone or tablet. Gesture Morpher's test environment runs on the device and loads the video frames related to the selected transformation and interaction. When Joan performs the interaction on the device, the environment responds with frame changes corresponding to the amount of change in the parameter values of the specific transformation. These are caused by the touch movement made by Joan, e.g., the movement in pixels for translation, change in angle for rotation, and change in size for scaling.

Joan quickly tests a few interaction options on an Android device before deciding on one, and then exports her interaction selections (along the video) as a Gesture Morpher design file. In a design meeting, Joan passes around the devices that have Gesture Morpher and the design file installed. Her teammates and customers are able to directly experience the multi-touch interaction she chose, and performing the interactions generates appropriate feedback on the interface. Subsequently, Joan acquires rich, constructive feedback on her design from peers, which might never have emerged had she solely presented static mockups of the UI.

Another situation that designers often encounter is adding new interaction behaviors to an existing or legacy application. For example, a designer might want to use a multi-touch interaction to adjust the text size of a note-taking app that is currently achieved by dragging a slider — the size of which is too small to acquire with a finger on a touch device. It is often difficult to alter the implementation of such an application for new input behaviors. Because a designer often needs to explore a range of interaction alternatives, the substantial effort and expertise required of modifying an existing implementation can significantly slow the interaction design process. Prior work such as Prefab [3] allows a user to easily enhance an existing interface; however, the prior work does not address multi-touch interaction behaviors as we discussed earlier.

With Gesture Morpher, designers can easily test a range of multi-touch interactions for a legacy application in the same way as they would for a UI mockup. For the scenario above, the designer can record a video of text size changing behaviors of the note-taking application. By feeding the video to Gesture Morpher, which detects the text-scaling transformations demonstrated in the video, the designer can try out a set of interactions to achieve the same effect, such as Pinch, Double-Tap or Tap-and-Pan. Once the designer and her team are happy with the interaction design, they can realize the design by modifying the existing implementation.

ALGORITHMS

There are two major tasks we need to perform: 1) from a video recording, extract UI behaviors as transformations applied to UI content, and 2) associate live interaction input with the extracted UI transformations such that the output responds appropriately for the user input.

Extracting UI Behaviors as Transformations

There are two steps to extract UI behaviors from a video recording: motion tracking to detect UI object movement over time, and transformation extraction to identify desired transformations manifested by moving objects, which determine appropriate interaction input. Before these steps are executed, a preprocessing stage identifies the relevant frames that contain significant movement.

Identifying Candidate Frames

Only a subset of the total available frames in a video contains movement relevant for processing transformations. Further, video encoders used for screen recording on mobile devices tend to introduce noise in the form of frame-jumps and repeated frames in the video. The following steps help remove

much of this noise from the video to ensure that only relevant frames of the video are processed further.

- If the frame is empty, it is discarded.
- For each pair of consecutive frames, if more than 5% of all pixels are same between the two frames, there is an absence of significant movement between them and the new frame is discarded. The 5% value accounts for noise introduced by video codecs.
- The frame rate value associated with each frame determines the duration for which the frame stays visible on screen. For a video running at 30fps, the expected duration is one-thirtieth of a second. If the duration of a frame exceeds this value by more than five times (i.e. is greater than 5/30 of a second), it is discarded.

Motion Tracking

We used a feature-tracking approach to extract motion between frames extracted from the preprocessing stage. For these frames, we employ the Shi-Tomasi’s algorithm [13] for identifying features. These features roughly correspond to the points with sufficiently high brightness gradient and tend to occur at corners and edges within an image. By tracking these features across frames, we can construct a local motion estimation model.

We assume that feature motion is near-linear within the frame interval. We use a Kalman filter to locate features from one frame to the next. To ensure tracking accuracy, a feature from the current frame that we locate in the next frame is subsequently reverse located in the current frame. We discard all features whose distance between the original and mirrored position exceeds 1 pixel. As we described earlier, the number of features that can be identified on a UI is often small. Furthermore, many of them are lost when the pixels they are tracking either move out of the screen or are occluded by other objects. As a result, we perform frequent re-identification, doing so whenever the number of features drops below 5.

Transformation Extraction

To extract transformations from detected feature motions, we segment the video into a sequence of *blocks*. A block is a set of frames that encapsulates a single continuous movement. To identify these blocks, we use a delimiter of 10 continuous frames of no feature-motion (including the frames discarded earlier). We empirically found that the 10-frame period effectively captures the period (roughly equivalent to one-third of a second) between the user ending one interaction and beginning another. For example, when interacting with a map view, a user might sequentially switch between using Pinch and Pan gestures. A 10 frame threshold represents the time period between her lifting a finger to complete pinching and beginning a drag with the other finger. Each of the two blocks thus identified represents a unique transformation that is easier to extract than a mix of them.

For each block, given a set of feature points in one frame, x , and the matched feature points in the subsequent frame, x' , we seek a transformation that can best transform x to x' , i.e., $x' = Tx$. We estimate T by fitting multiple 2D linear

transformation models and choose the one that results in the least fitting error. The transformation models we consider include those for translation, rotation, and similarity. Here, we refrain from fitting a general affine model that also considers other transformations such as shear, which are not commonly presents in 2D touchscreen interfaces.

We perform iterative fitting similar to weighted-least squares where each subsequent iteration uses the inverse of errors from the previous round as weights. The error is calculated as the difference of the actual position and the position estimated by the transformation generated in the previous iteration. This ensures that features that fit the model more are preferred over those that fit less. We perform this iterative fitting at most 10 times, resulting in a refined transformation between the two frames. Repeating this process for all pairs of adjacent frames generates a series of transformation matrices.

Based on the extracted transformation matrices, we acquire parameters for rotation, scaling or translation movement on the interface and determine the transformation type between each pair of frames. We then use the most frequently occurring transformation within the frames of a block as the transformation type for the block. Operating on blocks smoothens the extraction and effectively removes noise otherwise present in transformations extracted frame-wise.

Associating Interactions with Extracted UI Behaviors

Once we extract all the UI transformations in the video, we assign appropriate interactions for each transformation type. To do this, we created a touch-based interaction/gesture vocabulary (Table 1) that categorizes the most appropriate interactions for each transformation type based on the conceptual findings from previous work, e.g. [14]. The vocabulary is built on a

Visible Action	DOF	Interaction/Gesture
Select	1	One Finger TAP One Finger DRAG One Finger DOUBLE-TAP
Scale	2	Two Finger PINCH One Finger DOUBLE-TAP One Finger TAP AND PAN
Rotate	3	Two Finger ROTATE One Finger TAP AND PAN Two Finger PAN
Drag	2	One or Two Finger PAN
Scroll	2	One or Two Finger PAN
Next/Previous	1	One Finger PAN One Finger TAP
Hide/Show	1	One Finger TAP One Finger DOUBLE TAP

Table 1. Touch-based interaction vocabulary that utilizes the visible change in the pixels of a frame as the primary classifier. Only single-handed interactions are used since we focus primarily on handheld devices such as smartphones and tablets.

set of primitive touch-based interactions, including Tap, Pan, Pinch, Drag, and Rotate.

Frame Access In Response To Live Gesture Input

To allow a designer to test an interaction, we need to know which video frames to show in response to the live touch input. Given a starting frame, a naive method to determine if we should go to an adjacent frame is to check if the touch movement has exceeded the magnitude of movement required from the current frame to the adjacent one, as detected in the analysis stage.

However, this strategy has a few limitations. Moving the fingers too fast will generate a much larger magnitude of movement than the magnitude between two adjacent frames. Limiting the next frame to only adjacent frames will result in a jarring and unresponsive UI effect. More critically, the strategy fares poorly at the crests and troughs (or cusps) of movement. These occur when a motion in the video involves a change or inversion of direction, such as the user panning to the right and then back to the left. At these inversion frames, this strategy becomes unstable as both the previous and next frames expect the same type of touch movement. Dragicevic et al. [4] address this issue with the Directional Continuity requirement, stating that users following a cusp should not go back to where they came. Although the results were somewhat better with this solution, they were still ineffective at predicting user's intention and accurately mapping it to the response.

To overcome these challenges, we employed a more robust approach for accessing frames by recognizing the underlying anatomy of the interface. Consider a video recording that captures the following navigation on a mobile home screen: Starting at Screen 2, the user swipes leftwards twice and reaches Screen 4, swipes rightwards twice and returns to Screen 2, and finally swipes rightwards again and reaches Screen 1. In sum, the raw frames of the video manifest the UI screens as shown in Figure 2.

When testing a Swipe interaction based on this video, if we start from the first frame (depicting Screen 2) and only consider adjacent frames, swiping rightwards would generate no response because the only movement expected at the frame is to swipe leftwards, although in the actual interface the user would see Screen 1. To increase the degree of freedom in responding to user actions at test time, we extract a state transition diagram from a video recording, where each state corresponds a cluster of frames that represent the same screen and each transition captures the transformation that has to occur to transform the frames of one state to another. For example, by knowing that frame 1 and 5 represent the same screen, Screen 2, we can create a state that has two outgoing transitions and

Figure 2. A user navigates the home screen of a mobile phone using swipe gestures.

allows the user to swipe in both directions. (Note that a similar sequence folding strategy for forming a state transition diagram has been used previously in FrameWire [9]. However, we are addressing continuous UI changes while the previous work focused on discrete screen switching.) Based on the extracted state transition diagram, at a given state we determine the next state (and thus, next frame) to go to by finding the outgoing transition whose transformation is the closest match to interaction input.

If the user reaches an edge state while performing an interaction, with no further frames available in the direction of movement, we transform the current frame to respond to user input. In case of a scaling operation, the frame scales up or down; for rotation, the frame rotates around the z-axis; and for translation, the image rotates around one or both of X and Y-axes, similar to the conventional response of an Android UI for edge cases. The frame animates back to its original form when the user lifts his or her finger.

IMPLEMENTATION

We implemented Gesture Morpher in Java for both the desktop and the mobile components. We used the Android Debug Bridge (ADB) to communicate between the desktop and the device for live recording of app behaviors and quick testing of interactions. The input video can be a recording based on any operating system, but the Test environment currently only runs on an Android-powered device. We extensively used OpenCV for processing video frames.

DISCUSSION

The objective of Gesture Morpher is to support easy and rapid testing of multi-touch interactions by mapping these interactions to targeted behaviors extracted from video frames. It is particularly beneficial in situations when the designer needs to test among several alternatives. For instance, in drawing or sketching applications, since the default Pan (Drag) gesture is employed by the drawing action, an alternate operation is required for dragging the view. Relevant options are Two-Finger-Pan, Tap-and-Pan, and Hold-and-Pan (holding with one finger while panning with the other). With an input video demonstrating a view being dragging, a designer can use Gesture Morpher to rapidly test all three options directly on a device and identify the most appropriate one.

We demonstrated the feasibility of our approach by supporting an initial set of transformations and interactions that are applicable to a wide set of application behaviors. However, we also identified several limitations with our current approach. We discuss two of these below.

Limited Vocabulary

In the vocabulary presented in Table 1, we employ a limited set of everyday interactions, such as Tap, Pan, and Pinch. For our initial prototype, these interactions provide an adequate coverage for the extracted behaviors as most interactions we use across applications tend to be combinations of Tap, Pan, Pinch and Rotate. However, for more complex and feature-rich interfaces, the vocabulary may not provide adequate support. For instance, to address complexity in interfaces, researchers have leveraged properties of touch such as pressure and shear

[5] and used techniques such as touch overloading [6] to identify novel interaction techniques. There is a clear need for the vocabulary to be extensible to support these and other novel interactions so as to support more complex scenarios on top of our core idea. The challenge, however, is identifying a reasonable mapping from these interactions to the application behaviors.

Behavior Limitations

Since we deploy feature extraction algorithm on the input video, the interaction behaviors we extract are naturally restricted to geometric transformations. These include scaling, rotation and translation — behaviors with strong mappings to continuous interactions. For frames with multiple distinct movements, we identify the most prominent motion. This is because distinct movements controlled simultaneously by a user rarely appear in the wild. A more frequent case is of interactions that morph between modes (e.g., Rotate $\leftrightarrow$ Pinch). Identifying such interactions is a limitation of our system. The challenge here is to infer and interpolate the regions of existing frames to synthesize new frames that are in effect the correct response to user's actions, but don't exist in the original video.

CONCLUSION & FUTURE WORK

We presented Gesture Morpher, a tool for designers to easily test multi-touch interactions in the context of application behaviors, using synthesized output from video, without any programming, which is agnostic to the form of construction or implementation details of these behaviors. Our tool automatically detects and reconstructs dynamic continuous interaction behaviors within the video and associates multi-touch interactions that are appropriate for accomplishing these behaviors. A natural continuation of this work is to support other application behaviors beyond the transformations the system currently supports. Because creating a recording of dynamic behaviors for a UI mockup may require nontrivial effort, another direction is to extract dynamic behaviors from static images. However, our work takes a crucial first step in this direction and fosters our goal of reducing the effort of designing and testing multi-touch interaction.

REFERENCES

1. Nikola Banovic, Tovi Grossman, Justin Matejka, and George Fitzmaurice. 2012. Waken: Reverse Engineering Usage Information and Interface Structure from Software Videos. In *Proc. of ACM UIST 2012*. 83–92. DOI: <http://dx.doi.org/10.1145/2380116.2380129>
2. Michel Beaudouin-Lafon. 2000. Instrumental Interaction: An Interaction Model for Designing post-WIMP User Interfaces. In *Proc. of ACM SIGCHI 2000*. 446–453. DOI: <http://dx.doi.org/10.1145/332040.332473>
3. Morgan Dixon and James Fogarty. 2010. Prefab: Implementing Advanced Behaviors Using Pixel-based Reverse Engineering of Interface Structure. In *Proc. of ACM SIGCHI 2010*. 1525–1534. DOI: <http://dx.doi.org/10.1145/1753326.1753554>
4. Pierre Dragicevic, Gonzalo Ramos, Jacobo Bibliowicz, Derek Nowrouzezahrai, Ravin Balakrishnan, and Karan Singh. 2008. Video Browsing by Direct Manipulation. In *Proc. of ACM SIGCHI 2008*. 237–246. DOI: <http://dx.doi.org/10.1145/1357054.1357096>
5. Chris Harrison and Scott Hudson. 2012. Using Shear As a Supplemental Two-dimensional Input Channel for Rich Touchscreen Interaction. In *Proc. of ACM SIGCHI 2012*. 3149–3152. DOI: <http://dx.doi.org/10.1145/2207676.2208730>
6. Chris Harrison, Julia Schwarz, and Scott E. Hudson. 2011. TapSense: Enhancing Finger Interaction on Touch Surfaces. In *Proc. of ACM UIST 2011*. 627–636. DOI: <http://dx.doi.org/10.1145/2047196.2047279>
7. Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. 1985. Direct Manipulation Interfaces. *Human-Computer Interaction* 1, 4 (Dec. 1985), 311–338. DOI: http://dx.doi.org/10.1207/s15327051hci0104_2
8. Kenrick Kin, Bjorn Hartmann, Tony DeRose, and Maneesh Agrawala. 2012. Proton: Multitouch Gestures As Regular Expressions. In *Proc. of ACM SIGCHI 2012*. 2885–2894. DOI: <http://dx.doi.org/10.1145/2207676.2208694>
9. Yang Li, Xiang Cao, Katherine Everitt, Morgan Dixon, and James A. Landay. 2010. FrameWire: A Tool for Automatically Extracting Interaction Logic from Paper Prototyping Tests. In *Proc. of ACM SIGCHI 2010*. 503–512. DOI: <http://dx.doi.org/10.1145/1753326.1753401>
10. Yang Li, Hao Lu, and Haimo Zhang. 2014. Optimistic Programming of Touch Interaction. *ACM Trans. Comput.-Hum. Interact.* 21, 4 (Aug. 2014), 24:1–24:24. DOI: <http://dx.doi.org/10.1145/2631914>
11. Jun Rekimoto. 2002. SmartSkin: An Infrastructure for Freehand Manipulation on Interactive Surfaces. In *Proc. of ACM SIGCHI 2002*. 113–120. DOI: <http://dx.doi.org/10.1145/503376.503397>
12. Dennis Timothy Schaaf and Vineet Haresh Mehta. 2012. Interface for vehicle function control via a touch screen. (Jan. 2012). <http://www.google.com/patents/US8095278>
13. J. Shi and C. Tomasi. 1994. Good features to track. In , *1994 Proc. of IEEE CVPR '94*. 593–600. DOI: <http://dx.doi.org/10.1109/CVPR.1994.323794>
14. Jacob O. Wobbrock, Meredith Ringel Morris, and Andrew D. Wilson. 2009. User-defined Gestures for Surface Computing. In *Proc. of ACM SIGCHI 2009*. 1083–1092. DOI: <http://dx.doi.org/10.1145/1518701.1518866>
15. Mike Wu and Ravin Balakrishnan. 2003. Multi-finger and Whole Hand Gestural Interaction Techniques for Multi-user Tabletop Displays. In *Proc. of ACM UIST 2003*. 193–202. DOI: <http://dx.doi.org/10.1145/964696.964718>
16. Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. 2009. Sikuli: Using GUI Screenshots for Search and Automation. In *Proc. of ACM UIST 2009*. 183–192. DOI: <http://dx.doi.org/10.1145/1622176.1622213>