GUI Robots: Using Off-the-Shelf Robots as Tangible Input and Output Devices for Unmodified GUI Applications

Darren Guinness, Daniel Szafir, and Shaun K. Kane
University of Colorado Boulder
Boulder, CO, USA
{darren.guinness, daniel.szafir, shaun.kane}@colorado.edu

ABSTRACT
Traditional GUI applications provide limited support for tangible interaction, as most applications are not programmed to support tangible input, and most input devices do not provide haptic feedback. To address this limitation, we introduce GUI Robots, a software framework that enables developers to repurpose off-the-shelf robots as tangible input and haptic output devices, and to connect them to unmodified desktop applications. We introduce the GUI Robots framework and present several proof-of-concept applications, including a haptic scroll wheel, force feedback game controllers, a 3D mouse, and a self-driving notification robot. To evaluate whether GUI Robots can be used to prototype tangible interfaces for existing applications, we conducted a user study in which developers created customized tangible interfaces for two applications. Study participants were able to create tangible user interfaces for these applications in less than an hour. GUI Robots allows developers to easily extend applications with tangible input and haptic output.

Author Keywords
Tangible user interfaces; human-robot interaction; graphical user interfaces; end-user programming; tactile feedback.

ACM Classification Keywords
H.5.2. User interfaces: Input devices and strategies.

INTRODUCTION
Modern desktop applications often provide rich visual and audio experiences, supported by developments in high-resolution displays and high quality stereo audio. However, existing applications provide little support for tangible interaction or haptic feedback. Thus, the advantages of tangible interaction [13] remain unavailable in most desktop applications.

DIS 2017, June 10-14, 2017, Edinburgh, United Kingdom

ACM 978-1-4503-4922-2/17/06…$15.00
DOI: http://dx.doi.org/10.1145/3064663.3064706

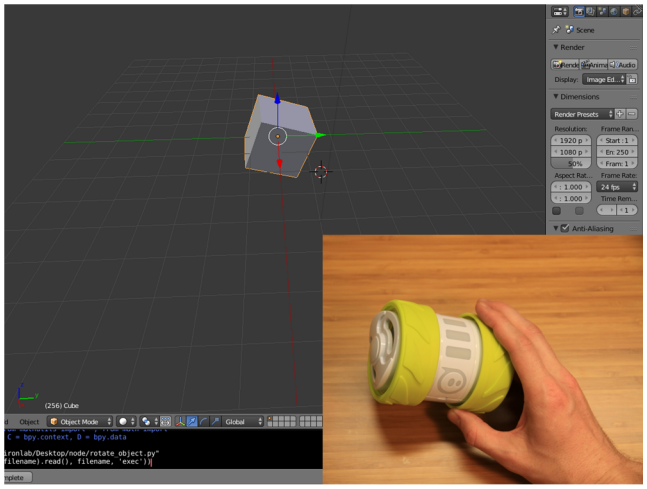

Figure 1. Using an off-the-shelf robot (Ollie) as a tangible input/output device for a 3D modeling application (Blender). Our system dynamically pairs the robot with the on-screen model: manipulating the robot moves the model; changing the model on screen causes mirrored movements in the robot.

The lack of tangible input and haptic feedback in desktop GUI applications is due to several factors. First, most applications are designed to support traditional input devices such as keyboards, mice, and touchpads, and these devices typically do not provide haptic feedback. While some input devices do provide force feedback, such as the Apple Magic Trackpad and some gaming mice, most applications do not support the advanced haptic features of these devices. Tangible interaction is also supported by some specialty input devices such as 3D mice and haptic game controllers, but these devices are mostly used by professional 3D designers and gamers, respectively, and are not used by most mainstream computer users. Second, adding haptic feedback to applications typically requires that developers add new features to their code, and few developers do so.

In this work, we explore opportunities for bringing tangible input and haptic output to desktop GUI applications by addressing these two challenges. First, we address the lack of tangible input devices by repurposing an existing technology, that of educational and toy robots such as Sphero, Wonder Workshop's Dash, and Parrot's AR Drone. These robots are inexpensive (often less than $100 USD) and support connecting to PCs via Bluetooth and Wi-Fi. Most importantly, these robots contain sensors such as

accelerometers and gyroscopes that enable them to be repurposed as input devices, and actuators that can be used to provide haptic feedback. Second, we address the lack of software support for tangible input and haptic output by enabling our tangible input devices to be paired with existing applications without changing the underlying code, through a combination of input event emulation, GUI automation, and custom application APIs.

Contributions

In this paper, we introduce the GUI Robots framework, which enables developers to add tangible input and haptic output to existing applications using off-the-shelf robot platforms. We introduce several proof-of-concept applications in which desktop GUI applications are augmented using robots as tangible input devices. Finally, we demonstrate that our framework enables developers to add tangible input and haptic output to existing applications through a user study in which 12 developers used our framework to create a tangible game controller and a tangible video editing tool. The contributions of this paper are:

1. The GUI Robots framework, which enables developers to author tangible input and haptic output behaviors using robots, and to attach these behaviors to existing applications;
2. Demonstrating the use off-the-shelf robots as an affordable and accessible approach to providing tangible input and haptic output;
3. Demonstrating how input event emulation and GUI automation can be used add haptic interaction to existing applications;
4. Proof-of-concept applications for extending GUI applications with tangible interaction;
5. Insights from a user study in which 12 developers used the GUI Robots framework to create their own tangible interactions for an existing video game and video editor.

RELATED WORK

Tangible User Interfaces in HCI

Our work draws inspiration from the visions of tangible computing brought forth by Sutherland [27], and by projects such as metaDESK [28], MagicDesk [25], and the Actuated Workbench [23]. These projects, and those that have followed (e.g., [11,29,30]), demonstrate how tangible interaction can improve user engagement and can support more natural interactions in 3D space. While these projects have demonstrated the benefits of tangible interaction, they have typically taken the form of self-contained systems. Our goal in this work is to extend the benefits of tangible user interfaces to a broad range of mainstream PC applications.

Other research has explored how traditional PC applications can be augmented through the use of specialized haptic devices. Miller and Zeleznik [20] explored adding haptic feedback to GUI applications using the Phantom haptic controller. Several projects have explored the creation of haptic mice that can provide haptic output while interacting with a GUI [1,4]. Bonfire [15] and Portico [3] used computer vision to support tangible interactions around a laptop and tablet, respectively. TeslaTouch [7] and TPad [21] provided haptic feedback for touch screen applications by augmenting the touch screen hardware with electrovibration and ultrasonic actuators, respectively. Our work extends prior work in this area by supporting haptic interactions via novel devices (off-the-shelf consumer robots), and by providing support for adding tangible input and haptic output to existing desktop applications.

Robots as Tangible User Interfaces

Robotic devices offer great potential for use in tangible computing, as they feature an embodied physical form, and can support a variety of physical input and output modes.

Several projects have explored the use of robots to display information. Jacobsson et al. [14] created a series of shape-changing displays using an array of educational robots, allowing users to control the display by moving or shaking robots. Alonso-Mora et al. [2] created a dynamic display made up of a swarm of color changing robots, in which each robot could move and change color to represent pixels in a source image or animation. Robert et al. [26] developed a system that supported interaction with game characters, in which robots representing characters could be controlled with a joystick. BitDrones [12] used flying quadcopter drones with attached LEDs and displays to represent information in 3D, and enabled users to physically move the drones to interact with the display. These projects used mobile robots as physical manifestations of an image, animation, or game, and did not fully explore the input capabilities of the robots, as we do here.

Other projects have explored the use of robots to provide tangible input for applications on touch screens. TouchBugs [22] and Tangible Bots [24] used custom robots to provide tangible interaction on touch screen-based tabletops. Users could directly interact with the robots, which passed this interaction data to the underlying application, and the robots could move themselves across the tabletop to present various input configurations. Our work extends this general approach by using robots to control desktop GUI applications, but leverages off-the-shelf robots rather than custom-made robots, and instruments existing applications in addition to supporting new applications.

Zooids [17] introduced the concept of swarm user interfaces, in which a collection of wheeled robots moved in coordination to create a tangible user interface that could be manipulated by the user. Zooids can be used to represent data on a table surface, or can enable interactive drawing. Both Zooids and GUI Robots allow control of applications by physically manipulating robots. However, our GUI Robots framework complements Zooids by exploring ways to instrument existing user interfaces with tangible robots. Furthermore, while Zooids make use of custom robots and an augmented workspace, our approach focuses on

leveraging off-the-shelf robots in an uninstrumented environment, potentially lowering barriers to use.

End-User Robot Programming

Several prior projects have explored end-user-friendly approaches to programming robotic behaviors. Phybots [16] is a software toolkit that enables users to program robots to perform activities around the home. Bartneck et al. [6] developed tools to enable users to create shape displays using robots. In contrast to these prior toolkits, the GUI Robots API allows developers to specify the robot's behaviors from the user's perspective, enabling developers to easily detect gestures performed on and around a robot, and to author haptic interactions for handheld robots. By allowing developers to specify behaviors from a user-centric interaction perspective, rather than a robot-centric perspective, the GUI Robots framework may allow developers to more easily integrate tangible input and haptic output into their applications.

GUI ROBOTS

The primary contribution of this work is GUI Robots, a software framework that supports pairing off-the-shelf robots with PCs, authoring tangible input and haptic output behaviors using these robots, and pairing these robots with unmodified GUI applications. The following section describes the various components of this framework.

Connecting PC Applications with Robots

The GUI Robots framework is a Node.js library that can be used on all major desktop computing platforms. Connections between the PC and robots use the robots' integrated Bluetooth or Wi-Fi connectivity.

The GUI Robots framework uses a plug-in architecture to enable connections to a variety of robot platforms. Developers can add support for additional robots by connecting to the robot via Cylon, Node.js, or WebSockets, and by implementing the core interactions of the GUI Robots framework via the robot's native API. In general, the GUI Robots framework should be capable of supporting most robots with Bluetooth or Wi-Fi connectivity, as long as the robot's native API offers the ability to track the robot's motion and to control the robot's movement. A robot's specific capabilities as an input device depend on the size, shape, and movement characteristics provided by the robot itself, so not all GUI Robots can support all forms of interaction. For example, only flying robots such as the AR Drone can lift themselves off of the ground, and only robots that provide back-drivable motors (i.e., robots in which a user can physically manipulate the motor's position) can provide haptic resistance while being moved by the user.

To date, we have tested our framework with four robot platforms: Sphero and Ollie (from Sphero, Inc.), Wonder Workshop's Dash, and Parrot's AR Drone. The present work has primarily used the Sphero and Ollie robots, as these robots provide a variety of sensors and actuators in a handheld form factor (Figure 2). Sphero is a spherical robot containing a self-balancing platform with two independently driven wheels, a 3-axis accelerometer, a gyroscope, a controllable RGB LED, and a Bluetooth radio. Ollie is a cylindrical robot that features a similar self-balancing mechanism, but uses wheels placed directly on the surface rather than rolling inside a ball. Because Ollie's wheels directly touch the surface, while Sphero's wheels are encased in a ball, Ollie can provide some forms of haptic feedback that Sphero cannot., such as pushing against the user's hand to provide friction while the user moves the robot. Both Sphero and Ollie were originally intended as educational toys, and cost approximately $100 USD each.

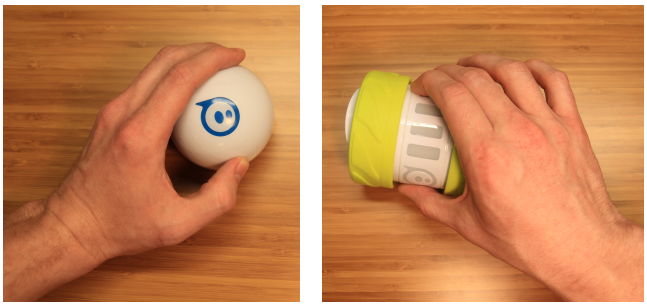

Figure 2. Sphero (left) and Ollie (right) robots.

Our framework uses the Sphero and Cylon.js APIs to control the Sphero and Ollie robots. While the Sphero API provides limited support for connecting the robot to a PC and reading the device sensors, the GUI Robots framework provides many additional features, including the ability to track user input events, author haptic behaviors, and connect to a variety of existing PC applications.

Recognizing User Input

GUI Robots facilitate user input by being directly manipulated and moved by the user, such as by physically moving the robot, rotating the robot, or whacking the robot, similar to prior motion-control-based systems. Sensor data is streamed from the robot to the PC. Developers can track the robot's raw sensor values, but they can also author interactions using event handlers for gestures such as *onWhack* and *onMoveLeft*.

The GUI Robots framework uses heuristic gesture recognizers to convert the robot's raw sensor data into input events. Many consumer-grade robots, including the robots we have tested with our prototype, use a gyroscope to track the orientation of the robot, and use an accelerometer to track movement through space (Sphero and Ollie also provide limited, but relatively inaccurate, odometry through the motors' back-EMF signal). While a gyroscope can provide reasonably accurate data about a robot's orientation, most accelerometers provide much less accurate data about a robot's movement and position. Because most accelerometers cannot provide accurate odometry data, the current version of our framework does not support directly tracking a robot's position. The accuracy of information about a robot's position is limited by the robot's sensors, and cannot be significantly improved without adding additional

sensors to a robot or augmenting the workspace with an overhead camera. We carefully considered these trade-offs during the design of our system, and decided that there was value in exploring the possibilities of interacting with unaltered off-the-shelf robots in an uninstrumented workspace. Thus, the current GUI Robots framework uses orientation-based events such as *onRotateRight* and movement-based events such as *onMoveLeft*, and does not fully support tracking a robot's position.

User input such as touching, tapping, and picking up a robot is currently detected via the gyroscope and accelerometer, as none of the robots we investigated supported capacitive touch input. However, by combining the limited sensors available in these consumer-grade robots, our framework allows developers to track a robot's rotation and movement, and to detect touch input from the user.

Haptic Output

GUI Robots support a variety of types of haptic feedback. By oscillating the motor, a robot can provide traditional force feedback methods such as haptic bumps and vibration. However, our framework also leverages the robots' ability to move themselves to provide additional forms of haptic feedback that are not found in many existing haptic devices.

Force Feedback

While our robots did not feature a dedicated vibration motor, GUI Robots can provide traditional force feedback by pulsing the device's motor and moving the robot while in the user's grasp. For example, a robot can quickly toggle its motors back and forth in each direction to create a haptic bump, and can increase the intensity of these motor movements to support varying levels of vibration.

This vibrational feedback, when combined with tracking the robot's movement, provides the additional capability of rendering haptic feedback within a specific area. For example, a robot might detect that it is being moved forward, and render a series of haptic vibrations while being moved, creating the sensation of a haptic texture on the surface beneath the robot.

Constrained Movement and Isometric Behavior

As GUI Robots are able to move themselves on a surface, they can provide additional haptic interactions by restricting or controlling their own movement. For example, a GUI Robot could be programmed such that it could be rotated in a clockwise direction, but not counterclockwise. As a user attempts to rotate the robot counterclockwise, the robot could resist that movement by actuating its motors to rotate clockwise. Although the user may be able to overpower the robot, the robot could still provide force feedback while being turned, and could rotate back to its preferred position once the user let go of the robot. Likewise, the robot could be programmed to move horizontally relative to the user's workspace, but not vertically, or could provide varying levels of resistance based on the direction being moved.

These haptic behaviors could be used to represent more complex tangible interactions. For example, an Ollie robot could represent the throttle in a flight simulation game. The robot could be programmed to support movement along the vertical axis only, matching the affordances of a physical throttle control. Additionally, the robot could provide greater resistance when being pushed forward than when being pushed backward, emulating the throttle's physical resistance while increasing speed. This constrained movement behavior is currently best supported by the Ollie robot, as it can easily push back against the user. Because the Sphero is encased in a ball, it cannot directly push back while being moved, although it can vibrate to provide force feedback, and can move itself back into its preferred position after being released.

In addition to constraining their own movement, GUI Robots can be programmed to represent isometric behavior. For example, if the robot detects being moved to the left, it can drive itself back toward its original position after it is released, emulating an isometric input device. While the positioning accuracy of our existing robots makes it difficult for the robot to return exactly to its origin, in practice we have found that small movements back toward the origin can demonstrate isometric behavior when the robot is used in a small space, such as on a desk surface.

Movement and Positioning

GUI Robots can move themselves independently to achieve a variety of objectives. For example, a robot can be driven in a specific pattern as a form of output, such as by drawing out a circle on the desk surface. This feature can be used to provide non-visual haptic output on the desktop. GUI Robots can also be programmed to move between pre-specified locations. For example, a user might store their GUI Robot in the corner of their desk. When the user activates a GUI Robot-enabled application, the robot could nudge itself from the corner toward the center of the desk to indicate that it is ready to be used.

Guiding the User via Dynamic Affordances

GUI Robots can leverage their movement capabilities to physically guide the user. For example, a user could place their hand on top of the robot, and the robot could drive itself in a specific direction or pattern, providing an additional dimension of haptic feedback. This mechanism could also be used to inform the user about how a tangible user interface works via a dynamic affordance [18]. For example, a map application might allow the user to zoom in and out by rotating the map, and roll the device forward to adjust the angle of the map view. These gestures might be difficult to discover for a novice user, but a GUI Robot could teach these gestures to a user by rotating slightly while simultaneously zooming the map on-screen, demonstrating how the movement of the robot maps to an action within the application.

Orientation and Movement	Input Events	Output Commands	GUI Automation
getRotation	onWhack	doBump	findImageOnScreen(imgName)
constrainRotation(dir)	onDoubleWhack	doShake	startTrackingImage(imgName)
constrainMovement(dir)	onShake	showColor(c)	stopTrackingImage(imgName)
onAccelerate(x,y,z)	onSpin(dir)	drive(dir)	onImageFound(imgName, x, y)
onRotate(roll,pitch,yaw)	onMove(direction)	constrainRotation(dir)	onMouseEvent(event)
	onPickUp	constrainMovement(dir)	onKeyEvent(event)
	onPutDown		doMouseEvent(event)
			doKeyboardEvent(event)

Table 1. Overview of GUI Robots API commands. Our API supports tracking orientation and movement events; receiving input events on the robot; providing haptic and visual output; and automating interaction with the GUI.

Mirroring Virtual Objects

GUI Robots can use their ability to position and orient themselves in order to "mirror" the status of an on-screen object. For example, a GUI Robot could be paired with a model in a 3D modeling application such as Blender (Figure 1). In this scenario, the user can move and rotate the robot to move and rotate the on-screen object. These movements could also be mirrored back from the on-screen application, so that if the on-screen model rotates, the GUI Robot could rotate itself to match (where permitted by gravity). This capability could be used to monitor information in an application, such as by tracking the movement of an opponent in an online game, or could be used to support collaborative tangible interactions with a remote colleague.

Connecting GUI Robots to Desktop Applications

A core feature of the GUI Robots framework is the ability to extend existing applications with tangible input and haptic output without needing to create a new application or alter a program's code. The GUI Robots framework supports three methods for connecting tangible robots to existing applications: emulating keyboard and mouse input, tracking and controlling UI elements with Sikuli [31], and using application APIs.

Emulating keyboard and mouse input: Many applications can be controlled simply by spoofing keyboard and mouse events. The GUI Robots framework can control an existing application by entering a sequence of keys or controlling the mouse cursor. This mouse emulation approach can be used to operate UI elements with a fixed location, such as clicking the Windows Start Button. More complex mouse interactions are possible using Sikuli, as described below. The haptic controller for Windows Movie Maker, described in the user study section of this paper, relies primarily on sending keyboard shortcuts to control the application.

Tracking UI elements with Sikuli: In many situations, a developer might wish to pair a robot with a specific GUI control. However, it can be difficult to automate interaction with an existing user interface, as many applications do not provide scriptable access to their GUI. Our framework provides the capability to locate specific UI elements on screen using computer vision, and to control them via keyboard and mouse events. Our framework uses Sikuli [31], which enables developers to capture a screenshot of an on-screen item, search for that item on screen, and interact with that item via emulated keyboard and mouse events.

This method can be used to instrument a variety of user interfaces across OS platforms, on the web, and even in games. For example, our Angry Birds controller, described in the next section, was created by tracking the on-screen bird image. When the user drags the robot backwards to simulate pulling back the slingshot, our application finds the bird on screen, clicks the mouse button, and drags the mouse down and to the left to pull back the slingshot.

Using application APIs: In some cases, a developer may wish to add features beyond what is possible through input emulation or Sikuli. If an application provides a public automation API, this API can be used to connect the application to the GUI Robots framework. For example, our 3D Mouse application is implemented using Blender's API, which provides precise information about the position and orientation of the on-screen 3D model. Similarly, the Haptic Web Browser Control uses a Chrome extension to read content on the current web page and to provide haptic feedback based on the page content.

GUI Robots Developer API

All input and output functions are exposed via a JavaScript API (Table 1). This API abstracts the connections to each robot, and offers a unified set of commands across robots, although not every robot supports every command. Where possible, commands are presented from the perspective of the user, rather than from the perspective of the robot, to simplify the development of tangible user interfaces. For example, the "move left" event uses the user's definition of left in their workspace, rather than the robot's left side. The API also allows access to a robot's low-level sensor data, enabling developers to create more advanced forms of input and output. The GUI Robots API also provides support for emulating keyboard and mouse events, and for tracking and interacting with GUI controls. A user evaluation of the API is presented later in this paper.

PROOF-OF-CONCEPT APPLICATIONS

Here we present several example applications, developed by our research team, that demonstrate the capabilities of our framework to extend existing GUI applications with tangible input and actuated haptic output.

Haptic Web Browser Control

We developed a plug-in for the Google Chrome browser that allows users to navigate web sites using a tangible robot controller. This plug-in also provides haptic feedback based on page content, demonstrating the capability of GUI Robots to provide both tangible input and haptic output (Figure 3).

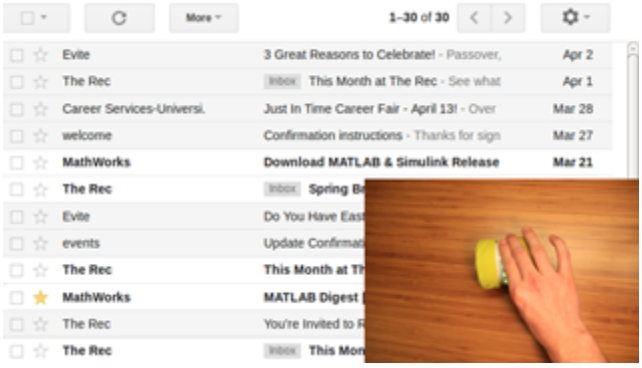

Figure 3. Haptic Web Browser Control enables the user to scroll a web page by rotating or rolling the robot, and provides haptic feedback based on the page content.

Using this application, the user can scroll through the current web page by spinning the robot or rolling the robot up and down. The robot can also be used as an isometric controller, so that moving the robot backwards (i.e., toward the user) will cause the web page to scroll downward continuously until the user moves it forward once again. Future versions of this application could support additional interactions using the robot's additional degrees of freedom. For example, rolling the robot forward and back could control scrolling, while twisting left and right could zoom in and out.

This application also supports dynamic haptic output based on the web page being viewed. Currently, the plug-in tracks top-level headers on the current web page (i.e., `h1` tags), and causes a haptic bump each time such a header scrolls into the current view. This feature allows the user to quickly scroll through a web page and to receive some haptic feedback about the structure of the page content. A stronger haptic bump is provided when the user scrolls past the top or bottom of the web page.

This haptic output capability can also be extended with site-specific scripts. For example, the current version of this application supports contextual feedback in Gmail: rotating the robot causes the cursor to step through each message, and generates a haptic bump when a starred message is selected (similar to the effect shown in [19]).

Force Feedback Game Controller

We have developed several prototypes to support tangible interaction with video games. These applications can support both motion input and haptic output..

For example, we have created a custom haptic controller for the game Angry Birds. As Angry Birds does not currently support desktop OSes, our controller uses a web-based clone that functions similarly to the original game. In this game, the user aims a slingshot containing a bird, and fires the bird toward a tower containing evil pigs. The user wins the game by knocking over each one of the pigs.

We developed a prototype of an Angry Birds controller using the GUI Robots framework and the Ollie robot (Figure 4). In our adaptation of the game, the user pulls back the slingshot by rolling the robot backwards. When this occurs, the GUI Robots framework identifies the location of the bird using Sikuli, clicks the bird, and drags the mouse cursor down and to the left. Once the slingshot has been pulled back, the user can rotate the robot to adjust the angle of the slingshot. The user can launch the bird by rolling or whacking the robot. Once the bird has been launched, the robot rolls forward a few inches and shakes back and forth, adding additional haptic feedback to the game. After a few seconds, the robot rolls back to its original position. This application demonstrates tangible input and haptic output, and shows that GUI Robots can control an existing game without access to the game's source code.

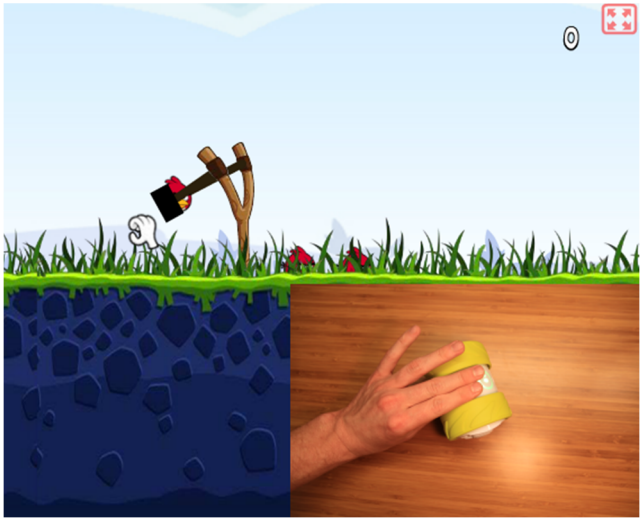

Figure 4. GUI Robot used as a tangible game controller. The user pulls the robot back to arm the slingshot. The robot launches itself forward and shakes after the slingshot is fired, adding expressive haptic output to the game experience.

We developed a similar controller for Valve's Counter-Strike, in which the user moves their character by rolling the robot, turns by rotating the robot, and fires by whacking the robot or pressing a mouse button. The robot vibrates when the user fires. In the future, this application could be extended to provide feedback based on specific in-game elements, such as vibrating when the player loses a match. More recently, we have created a prototype of a GUI Robot Input Manager (Figure 5), which allows the user to map a robot's movement to keyboard and mouse events. This tool could be used by non-programmers to create simple input mappings between GUI Robots and existing games.

3D Mouse with Object Mirroring

Manipulating 3D objects on a 2D screen can be difficult and cognitively demanding [10,25]. We developed a plug-in for

the 3D modeling tool Blender that enables the user to manipulate a model in 3D space by manipulating a paired GUI Robot (Figure 1). We used the Blender API to track the position and orientation of the model being edited. When the user wishes to manipulate the model, they may press a key to link the model to the GUI Robot. Once linked, moving and rotating the robot causes the corresponding 3D model to be moved. This pairing between the on-screen model and the GUI Robot is bi-directional, such that manipulating the on-screen model with a keyboard or mouse causes the paired robot to move and match the on-screen object (within the constraints of the robot's capabilities).

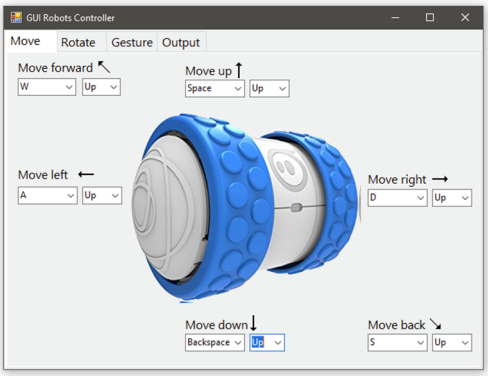

Figure 5. GUI Robot Input Manager allows end users to map interactions with the robot to keyboard and mouse commands.

Currently, this plug-in enables the user to manipulate the orientation and position of the object only; however, future versions of our system could enable richer interactions by further coupling the physical robot and virtual object. For example, tapping the robot's exterior could select a face of the 3D model or could swap between textures on that model.

Tangible Music Controller

Tangible input devices can be especially useful in the context of music and creative performance, enabling users to engage physically with a creative work. We developed a tangible music controller using the Max/MSP electronic music package (Figure 6). This application allows the user to control a music composition by manipulating the robot. In our prototype, rotating the robot, lifting it into the air, and whacking the robot each adjusted some parameter of the composition, such as the tone or tempo. During playback of an existing composition, the robot moves itself around the floor and flashes the device's LED lights in sync with the music. A performer or even an audience member could manipulate the music playback by blocking the robot's movement, or by moving the robot to a different location. In the future, this application could be used to create interactive multimedia performances that combine audio, visual, and tangible information.

Rolling Desktop Notification Manager

While most of our example applications explore how GUI Robots can be used as input devices, our framework also provides support for producing output via actuated tangible objects. To explore the possibilities of using a GUI Robot as an output device, we developed a notification manager application that allows the user to map specific notifications to behaviors from the robot.

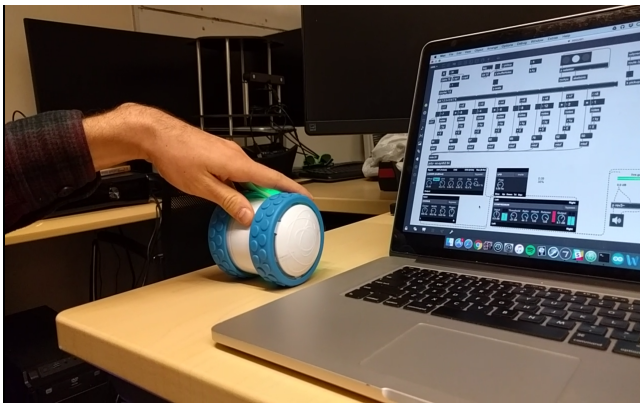

Figure 6. Ollie Robot paired to Max/MSP using the GUI Robots framework. The robot moves and flashes its lights based on the music. Manipulating the robot affects music playback.

Our Notification Manager prototype uses a Chrome plug-in to capture notifications delivered via the HTML Web Notification API, although in the future it could be extended to support platform-specific notification APIs. The user can specify a text pattern to listen for, and a series of robot actions to occur if a matching notification is received (Figure 7). For example, the user can specify that the robot should move left and then right if a notification mentioning "Facebook" is received. The user can snooze the notification by whacking the robot, or can pick up the robot to mute the notification.

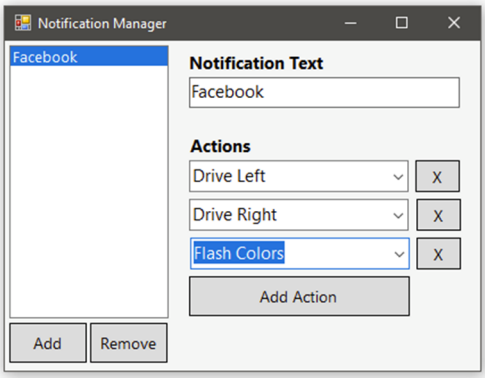

Figure 7. Notification Manager application enables a user to enter a notification pattern and specify a corresponding robot behavior. When a notification appears that matches the specified pattern, the robot performs the specified behaviors.

EVALUATING THE GUI ROBOTS FRAMEWORK

The GUI Robots framework is designed to enable the creation of tangible user interfaces that extend existing, uninstrumented applications. To evaluate the suitability of the current system for creating tangible user interfaces, we conducted a user study in which developers programmed two

tangible user interfaces for existing applications. Each participant took part in a single 90-minute session located in our lab. Participants used the Ollie robot for study tasks, as it provides the most flexibility for haptic output.

Participants

Twelve developers (9 male, 3 female, aged 22-37) participated in the study. Participants were recruited through university email lists and fliers placed on the university campus. Participants were not expected to have prior experience with programming robots, but were required to have prior JavaScript programming experience.

Procedure

The study comprised 5 phases: (1) introduction and consent; (2) GUI Robots framework tutorial; (3) Angry Birds task; (4) Movie Maker task; and (5) exit survey and debrief. The entire session lasted 90 minutes.

After completing the consent process, the participant was introduced to the goals of the study and the GUI Robots framework. Participants used a simplified version of the GUI Robots API presented in Table 1. Participants were given a copy of the framework source code, and given a cheat sheet that briefly described each of the API functions. The participant was given a tour of the WebStorm IDE, in which they were to write the study code, and given a brief tutorial on how to capture and track an image using Sikuli.

Next, participants were given 30 minutes to develop a haptic controller for the online Angry Birds clone. Participants were asked to first brainstorm and sketch out their solution, and then asked to develop their solution with the remaining time. During the programming task, a researcher observed the participant and answered questions about JavaScript and the APIs. Once the prototype was complete, the participant demonstrated the application for the researcher. The researcher then asked the participant how satisfied they were with their solution, and whether there were any changes or additions they would have liked to make to their prototype.

In the second task, participants were asked to make a tangible input device for controlling Windows Movie Maker, a video editor. This controller was expected to support the following commands: toggling play and pause, navigating to the previous and next clip, deleting the current clip, splitting the current clip, and undoing the last action. In addition, we asked developers to add haptic feedback when the video play head reached the 1-minute mark, and when the play head reached the end of the video. This activity followed the same structure as the previous activity.

Finally, the participant filled out an exit questionnaire that contained questions about each activity, and evaluated their experience using the System Usability Scale (SUS) [8]. Figure 8 shows a participant creating a prototype that used the Ollie robot.

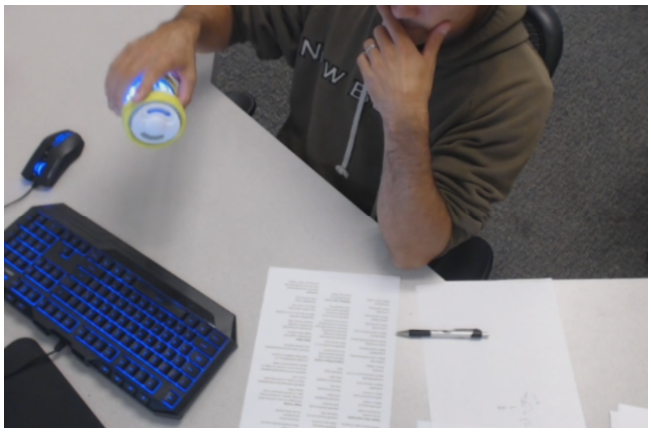

Figure 8. Study participant testing haptic feedback with the Ollie robot.

FINDINGS

All participants were able to successfully produce working versions of the haptic game controller. As we did not specify how to design the tangible interactions, participants developed a variety of solutions to solve this problem. Participants used several methods to aim and fire the bird, including rotating, moving, and whacking the robot. Some participants added haptic output after the bird was launched. Participants also added additional unrequested features, such as automatically clicking the on-screen restart button after the bird was launched, and using Sikuli to track the pigs' tower, and using the location of the tower to automatically aim the bird for maximum damage. Table 2 presents an overview of the approaches used to design the haptic game controller.

All participants also created a design for the Windows Movie Maker controller, but not all participants were able to implement all features in the 30-minute session. This task required developers to map six different actions to the movement of the robot, and sometimes participants chose mappings that conflicted with one another, causing slowdowns as they tested their solution. This task also required participants to use Sikuli to identify when the video play head was at 1 minute left, and at the end of the video, in order to provide haptic feedback, and several participants encountered difficulties capturing usable screenshots for Sikuli. In these situations, participants most often created a screenshot that was too small or too large, causing false positives in their haptic feedback. All participants except one (P9) were able to implement their complete solution; P9 completed the input methods, but did not finish the haptic output features. Participants were occasionally slowed down by connectivity errors with the PC's Bluetooth driver, which was corrected by a software update after the study session.

To get a sense of participants' experience using the GUI Robots framework, we asked participants to fill out the System Usability Scale (SUS) at the end of the activity [8]. This instrument features a set of 5-point Likert responses; a score of 70 is considered to be average [5]. Participant ratings

	Angry Birds				Movie Maker					
P#	Start	Distance	Angle	Fire	Play	Next/Prev	Delete	Split	Undo	Feedback
1	Whack	Move back	Rotate Y	Move back	Spin right	Move fwd/back	Double whack	Whack	Pick up	doShake, showColor
2	Accel X	Accel X	N/A	Move forward	Whack	Move fwd/back	Shake	Double whack	Move left	doShake, showColor
3	Put down	Move back	Spin	Pickup	Whack	Spin right/left	Double whack	Pickup/ Put down	Shake	showColor, driveLeft
4	Whack	Move fwd/back	Spin	Whack	Whack	Spin right/left	Pickup/ put down	Double whack	Move fwd	showColor
5	Whack	Whack	Yaw rotation	Double whack	Whack	Move fwd/back	Spin left	Double whack	Spin right	showColor
6	Move back	Move back	Pitch rotation	Move fwd	Move fwd	Yaw rotation	Shake	Double whack	Pitch rotate	doShake, doBump
7	Whack	Move fwd/back	Spin	Whack	Pickup	Move right/left	Double whack	Whack	Spin left	doShake
8	Whack	Move left	Move left	Double whack	Whack	Move right/left	Double whack	Pickup	Spin left	doBump, driveBack
9	Move fwd	Move left/back	Move left/back	Move left/ back	Whack	N/A	N/A	Image Found	N/A	doShake
10	Whack	Move left/right	Move fwd/back	Double whack	Whack	Move right/left	Double whack	Pickup	Shake	Drive forward
11	Move right	Shake	Shake	Shake	Whack	Move fwd/back	Double whack	Pickup	Put down	showColor
12	Pick Up	Move left	Move left	Whack	Whack	Spin right/left	Double whack	Pickup	Shake	doShake

Table 2. Study participants' input and output mappings for the two study tasks.

ranged from 62.5 to 87.5, with a mean score of 74.1 (SD=7.76). Three developers rated the system as above 80, and only two rated the system below the average, each at 62.5. While these scores suggest that there is room to improve the usability of our framework, it is not surprising that some participants found it difficult to learn a new technology in a short amount of time. Additionally, since participants were testing an early prototype of the system, it seems likely that satisfaction will improve with further testing and refinement.

Overall, we found that participants were engaged throughout the task, and expressed interest in using this framework in the future. Participants suggested a number of improvements to the tool, such as providing more granular control over gesture recognition (e.g., being able to set gesture thresholds or disable certain gestures) and providing more precise tracking of the robots. Participants were also interested in augmenting the existing robots with additional components, such as more precise positioning sensors, touch sensors and other buttons, and proximity sensors to detect obstacles and prevent the robot from rolling off the table.

Participants also expressed interest in using the API in their own personal projects. When asked about how they might use this framework in the future, P3 suggested that "controlling web browsers could be neat (moving the robot left and right to scroll through links, up and down to scroll the page, e.g.)"; P4 expressed interest in using the robot as part of Simon Says and other game; and P10 expressed interest in using the robot to notify him of events such as the end of a TV commercial, or when someone arrives at the door. These comments suggest that participants were able to consider uses of this framework beyond the study task; participants' suggestions also offer compelling ideas for future applications that use the GUI Robots framework.

DISCUSSION

The GUI Robots framework enables developers to repurpose off-the-shelf robots as tangible input and haptic output devices, and to connect these devices to existing applications to create new tangible user interfaces. Our framework supports connections with a variety of robots, and is easily extensible to add new robots. Our proof-of-concept applications show that the current framework can be applied to a variety of application areas, and can support the creation of diverse interactions, including game controllers, musical instruments, and ambient notification displays.

The goals of our user evaluation were to explore whether participants were able to create tangible user interfaces using our framework, to understand challenges to creating these user interfaces, and to explore further applications of this approach. By these metrics, our study was successful, in that all participants were able to create tangible controllers for two applications in a one-hour period, and demonstrated diverse approaches to creating tangible user interfaces, even for the same task.

We were pleased to see that participants did not create identical solutions, and that they were able to brainstorm and implement their own solutions to the problems. Participants often displayed creativity when describing the solutions they

came up with. One participant used the pick-up and put-down gestures to split clips in Windows Movie Maker, and justified his response by saying, "If I pick it up and put it down it will split the clip, kind of like a cleaver." When developing the controller for Angry Birds, P2 stated, "I would want to think of it as a big lever, whack it to pull it back, and twist it to change the angle, and whack it again to release." When testing different gestures, P6 stated that "spinning the robot seems to be the functional equivalent of using left and right the arrow keys". Participants created tangible input mappings using a variety of strategies, including leveraging metaphors to GUIs and physical controls. Participants were able to prototype and test tangible UIs even within the short study period.

Several participants noted accuracy issues with the gesture recognizers, as they experienced false positive and negative gesture events during the study. These errors are due in part to the relatively low accuracy of accelerometer-based motion tracking, and because the current framework uses simple heuristic methods for detecting motion gestures. In the future, the system's gesture recognition accuracy might be improved by collecting more training data for each gesture, or by introducing improved gesture recognition algorithms.

Participants also experienced some challenges when designing sets of multiple gestures, as gesture mappings sometimes interfered with each other. For example, performing the shake gesture could also trigger a pick-up gesture. In the future, the GUI Robots framework could provide suggestions to help users create non-conflicting gesture sets, or could adapt its gesture recognition based on the current gesture set.

Finally, participants experienced some challenges in using Sikuli to track elements on the screen. For example, in the Windows Movie Maker task, participants were asked to track when the play head reached specific points in the video. Successfully completing this task required participants to take a screenshot that would correctly identify when the video was at its end, but that would not cause false positives at other points during the video. Participants sometimes experienced false positives when testing this feature because they took screenshots that were too small or too large. This problem was exacerbated by Sikuli's fuzzy matching framework, which sometimes generated false positives. These problems could be minimized by providing developers with control of the detection thresholds, or by providing more comprehensive documentation on creating screenshots and using the Sikuli library.

LIMITATIONS AND FUTURE WORK

The present work presents new opportunities for tangible interaction, enabling developers to create new tangible user interfaces for existing applications. However, there are several areas in which this work could be extended.

First, the robots used in this study had limited features, and were limited in their ability to accurately track movement. Imprecise tracking caused gesture recognition errors, and prevented the use of some forms of interaction that would require precise tracking, such as using a robot as a stylus in a drawing application. We consider the capability to use existing off-the-shelf robots as tangible input devices to be a primary contribution of this work, and felt that the reduced accuracy was justified by the lower cost and increased availability of off-the-shelf robots. However, several study participants requested that we add additional capabilities to our robots, including more precise control over positioning, touch sensors, and other physical controls. In the future, we could extend the GUI Robots framework to cover a wider range of robots, or even to enable users to construct their own robots using Arduino or similar toolkits.

A second limitation of the current work is that it currently supports interaction with only one robot at a time. Future versions of the framework could support pairing multiple robots to a single application, enabling bimanual interaction. Given a set of available robots, a future version of the GUI Robots framework could suggest the best available robot for a specific task, for example suggesting a puck-shaped robot for controlling system volume.

A third limitation of the present work is that our evaluation targeted novice developers. While we believe that this population was optimal for evaluating the usability and versatility of the GUI Robots framework, we could extend the reach of this framework in both directions. On one hand, we could develop user-friendly tools that would enable non-programmers to create their own tangible user interface mappings. On the other hand, we could explore a more advanced version of the framework that supported more precise control over gesture recognition, and provided more robust features for controlling robots and instrumenting existing user interfaces.

CONCLUSION

In this paper, we presented GUI Robots, a software framework that enables the creation of tangible user interfaces for existing, unmodified GUI applications. Rather than developing custom robot hardware, GUI Robots repurposes off-the-shelf consumer robots as versatile and easily accessible tangible input devices. Our proof-of-concept applications demonstrate that this approach can be used to create a variety of tangible user interfaces, and can be used to enhance a wide range of existing software applications, including web browsers, 3D modeling tools, and video games. Our user study demonstrated that developers can use our framework to quickly prototype tangible interactions and attach them to existing applications. As consumer-oriented, wirelessly-connected robots become more ubiquitous, we hope this work can enable the creation of new user experiences, in which traditional GUI applications are extended by an ecosystem of helpful robots.

ACKNOWLEDGMENTS

We thank Willie Payne, Bryan Bo Cao, and Michael Walker for their assistance with this project.

REFERENCES

1. Motoyuki Akamatsu, and Sigeru Sato. 1994. A multi-modal mouse with tactile and force feedback. *International Journal of Human-Computer Studies*, 40(3), 443-453.
2. Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Roland Siegwart, and Paul Beardsley. 2012. Image and animation display with multiple mobile robots. *International Journal of Robotics Research*, 31(6), 753-773.
3. Daniel Avrahami, Jacob O. Wobbrock, and Shahram Izadi. 2011. Portico: tangible interaction on and around a tablet. In *Proceedings of the 24th Annual ACM symposium on User interface software and technology* (UIST '11). ACM, New York, NY, USA, 347-356. DOI: http://dx.doi.org/10.1145/2047196.2047241
4. Ravin Balakrishnan, Thomas Baudel, Gordon Kurtenbach, and George Fitzmaurice. 1997. The Rockin'Mouse: integral 3D manipulation on a plane. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (CHI '97). ACM, New York, NY, USA, 311-318. DOI: http://dx.doi.org/10.1145/258549.258778
5. Aaron Bangor, Philip T. Kortum, and James T. Miller. 2008. An empirical evaluation of the system usability scale. *International Journal of Human–Computer Interaction*, 24(6), 574-594.
6. Christoph Bartneck, Marius Soucy, Kevin Fleuret, and Eduardo B. Sandoval. 2015. The robot engine—Making the unity 3D game engine work for HRI. In *Proceedings of the 24th IEEE International Symposium on Robot and Human Interactive Communication* (RO-MAN), Kobe, 2015, 431-437. DOI: 10.1109/ROMAN.2015.7333561
7. Olivier Bau, Ivan Poupyrev, Ali Israr, and Chris Harrison. 2010. TeslaTouch: electrovibration for touch surfaces. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology* (UIST '10). ACM, New York, NY, USA, 283-292. DOI: http://dx.doi.org/10.1145/1866029.1866074
8. John Brooke. 1996. SUS-A quick and dirty usability scale. *Usability Evaluation in Industry*, 189(194), 4-7.
9. Morgan Dixon and James Fogarty. 2010. Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure. *In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '10). ACM, New York, NY, USA, 1525-1534. DOI: http://dx.doi.org/10.1145/1753326.1753554
10. George Fitzmaurice, Justin Matejka, Igor Mordatch, Azam Khan, and Gordon Kurtenbach. 2008. Safe 3D navigation. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games* (I3D '08). ACM, New York, NY, USA, 7-15. DOI: https://doi.org/10.1145/1342250.1342252
11. Sean Follmer, Daniel Leithinger, Alex Olwal, Akimitsu Hogge, and Hiroshi Ishii. 2013. inFORM: dynamic physical affordances and constraints through shape and object actuation. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology* (UIST '13). ACM, New York, NY, USA, 417-426. DOI: http://dx.doi.org/10.1145/2501988.2502032
12. Antonio Gomes, Calvin Rubens, Sean Braley, and Roel Vertegaal. 2016. BitDrones: towards using 3D nanocopter displays as interactive self-levitating programmable matter. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (CHI '16). ACM, New York, NY, USA, 770-780. DOI: https://doi.org/10.1145/2858036.2858519
13. Hiroshi Ishii and Brygg Ullmer. 1997. Tangible bits: towards seamless interfaces between people, bits and atoms. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (CHI '97). ACM, New York, NY, USA, 234-241. DOI: http://dx.doi.org/10.1145/258549.258715
14. Mattias Jacobsson, Ylva Fernaeus, and Lars Erik Holmquist. 2008. Glowbots: designing and implementing engaging human-robot interaction. *Journal of Physical Agents*, 2(2), pp.51-60.
15. Shaun K. Kane, Daniel Avrahami, Jacob O. Wobbrock, Beverly Harrison, Adam D. Rea, Matthai Philipose, and Anthony LaMarca. 2009. Bonfire: a nomadic system for hybrid laptop-tabletop interaction. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology* (UIST '09). ACM, New York, NY, USA, 129-138. DOI: http://dx.doi.org/10.1145/1622176.1622202
16. Jun Kato, Daisuke Sakamoto, and Takeo Igarashi. 2012. Phybots: a toolkit for making robotic things. In *Proceedings of the Designing Interactive Systems Conference* (DIS '12). ACM, New York, NY, USA, 248-257. DOI: http://dx.doi.org/10.1145/2317956.2317996
17. Mathieu Le Goc, Lawrence H. Kim, Ali Parsaei, Jean-Daniel Fekete, Pierre Dragicevic, and Sean Follmer. 2016. Zooids: building blocks for swarm user interfaces. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (UIST '16). ACM, New York, NY, USA, 97-109. DOI: https://doi.org/10.1145/2984511.2984547
18. Pedro Lopes, Patrik Jonell, and Patrick Baudisch. 2015. Affordance++: allowing objects to communicate dynamic use. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*

(CHI '15). ACM, New York, NY, USA, 2515-2524. DOI: https://doi.org/10.1145/2702123.2702128

19. Joseph Luk, Jerome Pasquero, Shannon Little, Karon MacLean, Vincent Levesque, and Vincent Hayward. 2006. A role for haptics in mobile interaction: initial design using a handheld tactile display prototype. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '06), Rebecca Grinter, Thomas Rodden, Paul Aoki, Ed Cutrell, Robin Jeffries, and Gary Olson (Eds.). ACM, New York, NY, USA, 171-180. DOI: http://dx.doi.org/10.1145/1124772.1124800
20. Timothy Miller and Robert Zeleznik. 1998. An insidious haptic invasion: adding force feedback to the X desktop. In *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology* (UIST '98). ACM, New York, NY, USA, 59-64. DOI: http://dx.doi.org/10.1145/288392.288573
21. Joe Mullenbach, Craig Shultz, Anne Marie Piper, Michael Peshkin, and J. Edward Colgate. 2013. Surface haptic interactions with a TPad tablet. In *Proceedings of the Adjunct Publication of the 26th Annual ACM Symposium on User Interface Software and Technology* (UIST '13 Adjunct). ACM, New York, NY, USA, 7-8. DOI: http://dx.doi.org/10.1145/2508468.2514929
22. Diana Nowacka, Karim Ladha, Nils Y. Hammerla, Daniel Jackson, Cassim Ladha, Enrico Rukzio, and Patrick Olivier. 2013. TouchBugs: actuated tangibles on multi-touch tables. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '13). ACM, New York, NY, USA, 759-762. DOI: https://doi.org/10.1145/2470654.2470761
23. Gian Pangaro, Dan Maynes-Aminzade, and Hiroshi Ishii. 2002. The Actuated Workbench: computer-controlled actuation in tabletop tangible interfaces. In *Proceedings of the 15th Annual ACM Symposium on User interface software and technology* (UIST '02). ACM, New York, NY, USA, 181-190. DOI: http://dx.doi.org/10.1145/571985.572011
24. Esben W. Pedersen and Kasper Hornbæk. 2011. Tangible Bots: interaction with active tangibles in tabletop interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '11). ACM, New York, NY, USA, 2975-2984. DOI: http://dx.doi.org/10.1145/1978942.1979384
25. Holger Regenbrecht, Gregory Baratoff, and Michael Wagner. 2001. A tangible AR desktop environment. *Computers & Graphics*, 25(5), 755-763.
26. David Robert, Ryan Wistorrt, Jesse Gray, and Cynthia Breazeal. 2010. Exploring mixed reality robot gaming. *In Proceedings of the fifth international conference on Tangible, embedded, and embodied interaction* (TEI '11). ACM, New York, NY, USA, 125-128. DOI: http://dx.doi.org/10.1145/1935701.1935726
27. Ivan E. Sutherland. 1965. The ultimate display. *Multimedia: From Wagner to virtual reality*.
28. Brygg Ullmer and Hiroshi Ishii. 1997. The metaDESK: models and prototypes for tangible user interfaces. In *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology* (UIST '97). ACM, New York, NY, USA, 223-232. DOI: http://dx.doi.org/10.1145/263407.263551
29. Malte Weiss, Julie Wagner, Yvonne Jansen, Roger Jennings, Ramsin Khoshabeh, James D. Hollan, and Jan Borchers. 2009. SLAP Widgets: bridging the gap between virtual and physical controls on tabletops. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '09). ACM, New York, NY, USA, 481-490. DOI: http://dx.doi.org/10.1145/1518701.1518779
30. Malte Weiss, Florian Schwarz, Simon Jakubowski, and Jan Borchers. 2010. Madgets: actuating widgets on interactive tabletops. In *Proceedings of the 23nd Annual ACM Symposium on User interface Software and Technology* (UIST '10). ACM, New York, NY, USA, 293-302. DOI: http://dx.doi.org/10.1145/1866029.1866075
31. Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. 2009. Sikuli: using GUI screenshots for search and automation. In *Proceedings of the 22nd annual ACM Symposium on User Interface Software and Technology* (UIST '09). ACM, New York, NY, USA, 183-192. DOI: http://dx.doi.org/10.1145/1622176.1622213