Pick-and-Play: A Cross-Device Asynchronous Input Framework

Paper id: 535

ABSTRACT

This paper proposes *Pick-and-Play*, a framework that helps users accomplish tasks by asynchronously migrating *user input actions* across devices. Unlike previous works that synchronize the application states or file contents between devices, *Pick-and-Play* is an asynchronous action-based framework. When using *Pick-and-Play*, the user *picks* the photo shot of a remote device with a mobile phone camera and performs inputs locally on the phone. The inputs are then automatically transformed into *play-macros* that can be replayed on the remote machine. The *play-macro* consists of a series of touch input coordinates and a homography matrix describing the coordinate mapping between the mobile display and remote display. By replaying the *play-macro* at different speed asynchronously, the *Pick-and-Play* framework can handle various practical scenarios such as the daily tasks simplification, sensitive data protection, control precision enhancement and control remapping. A technical evaluation shows that Pick-and-Play works robustly within the viewing angle of around 63° in front of the target display. A quantitative evaluation shows that there is no significant difference on the input performance between *Pick-and-Play* and native applications on mobile device. It suggested that *Pick-and-Play* could serve as a reasonable input alternative for targeting scenarios.

Author Keywords

Automation, camera, macro, multi-device environment

ACM Classification Keywords

H.5.2. User Interfaces: Interaction Styles

INTRODUCTION

Personal computing nowadays incorporates multiple heterogeneous devices. A single user could carry out daily tasks across multiple different devices according to the context of current environment and activity [10, 16].

The trend of increasing personal device numbers led to many research prototypes for cross-device interaction, e.g. [3, 6, 9, 13, 18, 19, 21]. In particular, the seminal Pick-and-Drop project [21] pioneered the pen-based direct manipulation technique for data transfer, Touch Projector [6] enabled users to interact with remote displays through live video on the personal mobile device, and Deep Shot [9] presented a framework that allow users to capture and transfer the application states across devices.

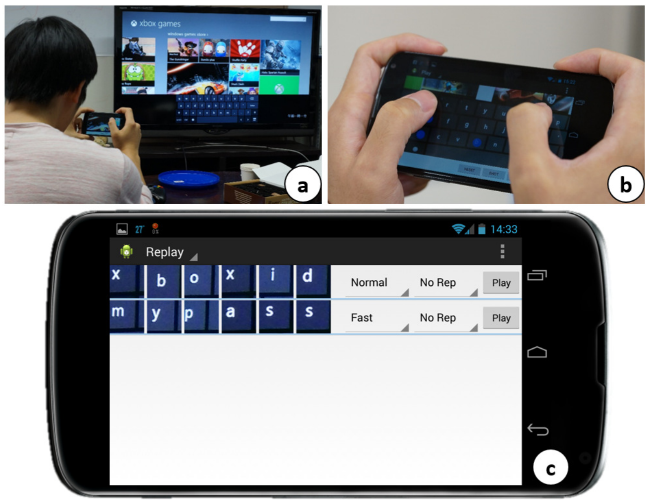


Figure 1. Pick-and-Play. (a) the user *picked* the screenshot of the remote display and (b) performed the inputs on the local device. (c) the inputs are transformed into a list of *play-macros* for asynchronous *replay.*

Overall, these previous works either focus on synchronizing the software states among devices [9, 18, 21] or on synchronously redirecting the user input from one device to another [6, 13, 19, 20, 27]. However, synchronizing software states requires specific support from the software side [9, 21], i.e. each application has to support the proposed protocol, while the synchronous input redirection might involve expensive video processing procedures [6, 19] or additional hardware setup [13, 20].

Complementing these previous efforts, we propose *Pick-and-Play*, a framework focusing on user actions, and explore the design space of the asynchronous cross-device interaction. The light-weight action-based framework requires only access to the input system on the OS level and the permission to capture and transfer the remote screenshot. *Pick-and-Play* directly works with existing applications while its asynchronous characteristics remove the requirement of expensive video processing.

Submitted to CHI 2014

We describe the interaction flow of *Pick-and-Play* through a real-world scenario (Figure 1). A user was playing a video game with joystick and at some point had to input the account information. However, the default joystick input was slow for text input [25]. With *Pick-and-Play*, the user first *picked* the on-screen keyboard with his phone camera (Figure 1a) and input the account information with the touch inputs (Figure 1b). The input sequences were then transformed into a list of reusable *play-macros* (Figure 1c). Afterward, the user could easily log into his account by replaying the recorded *play-macros*. In this scenario, *Pick-and-Play* brought the touch inputs to a non-interactive display, and enabled the user to transform daily repeating input actions into reusable personal macros.

The core component of *Pick-and-Play* is the *play-macro* that can be replayed and migrated between devices. Each *play-macro* consists of a series of touch input coordinates on the local display and a homography matrix, calculated by computer vision algorithms after *picking* the remote screenshot, describing the coordinate mapping between the local and remote displays.

Beyond the previous example, the *Pick-and-Play* provides four unique *play-macro* playback options that can help users accomplish tasks in various practical scenarios, such as daily tasks simplification, sensitive data protection, control precision enhancement and control remapping

A technical evaluation shows that Pick-and-Play works robustly within the viewing angle of around 63° in front of the target display. A quantitative evaluation suggests that *Pick-and-Play* has the similar input performance as the native input methods on the mobile device and could serve as a reasonable input alternative for targeting scenarios.

RELATED WORKS

Pick-and-Play is largely inspired by previous works about cross-device interaction and interaction at a distance.

Cross-Device Information Migration

Previous research projects explored the design space of migrating information across multiple devices. Pick-and-Drop [21] explored the interaction scheme where users could drag and drop the objects among different devices. Remote Clip [18] synchronized the clipboard among devices. WinCuts [26] allowed multiple users to share and interact with partial viewport of the window. While the PIE [20] system focused more on low-level mechanism for communicating between devices. Shoot&Copy [5] and Deep Shot [9] shared the information between devices based on the visual features of files and contents. In particular, Deep Shot migrated run-time states and users could resume the state of the application across devices.

Unlike previous works that transfer application states or file contents among devices, *Pick-and-Play* migrates users' input actions and can handle various different practical scenarios. The *Pick-and-Play* framework also has the benefit of being able to directly work with existing applications.

Interaction at a Distance

Many research prototypes explored the interaction of controlling virtual or physical objects at a distance. Object-Oriented Video pioneered the interaction of remote controlling machines in industrial plants through live video [27]. Its interaction was further extended in the Touch Projector project [6] with manual and automatic zoom and temporary freeze techniques. TouchMe system [11] tele-operated the pose and position of a robot through the third-person view video on a touch panel. Both Sketch-and-Go [22] and exTouch [14] systems provided direct spatial control over physical actuated objects or robots through live view video on a mobile device.

Some previous researches focused on using mobile device to access control and/or share contents on remote large displays. Pears et al. enabled absolute pointing with a smart phone by registering both phone and remote public display [19]. Ballagas et al. [2] introduced the Point & Shoot technique with which users could select an object by taking its photo and the Sweep technique that translated the mobile device movement to the virtual object movement. Virtual projection [3] projected the information onto a digital surface using the optical projection metaphor.

Pick-and-Play shares the two phase interaction paradigm, i.e. taking a picture then manipulating, with many previous works [2, 3, 5, 9, 19]. Yet our framework is unique for its ability to asynchronously replay the *play-macros* in different speed for different scenarios.

USER INTERFACE

We built a prototype system on the Android platform. The user interface of *Pick-and-Play* consists of two main pages: **play page** and **replay page**. Note that our prototype assumes an established persistent connection between the local mobile device and the remote machine.

Play Page

In the play page (Figure 2), the user can capture the photo and record the inputs. Clicking the SHOT button invokes the camera application on the mobile device for capturing the screenshot of the remote device, which could be a television screen, a desktop monitor or a large display. The user can then manipulate the viewport, e.g. zoom-in and translate, for easier inputs. When the REC button is toggled on, our system starts recording users' input actions.

Unlike the real user interface, the captured photo is static and does not respond to the input events. To help users better perceive what has been recorded, we visualize the recorded inputs as an overlay on the captured photo. The blue dots represent the clicking inputs and the blue lines represent the dragging gestures.

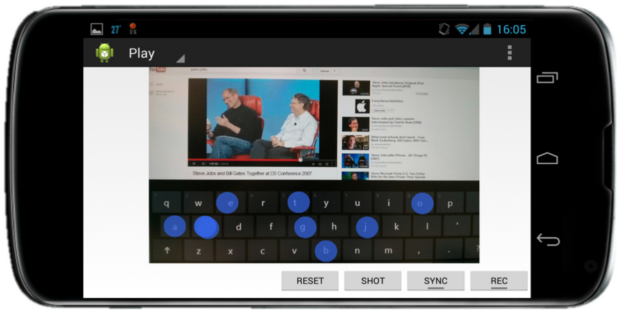


Figure 2. Play page. In this page, the user can invoke the camera app and record the play-macro. Blue dots are clicking inputs and arrows are dragging inputs.

Replay Page

Pick-and-Play segments recorded inputs based on the toggle of the REC button and translates them into a list of *play-macros* (Figure 3). At this page, users can configure how and when to replay the macros on the remote device. In particular, Figure 3a is the representative thumbnail image of the macro, copied from the captured photo according to the bounding box of the inputs. Figure 3b is the drop-down box for selecting macro replay modes, Figure 3c is the drop-down box for setting the interval between repeating replay, Figure 3d is the play button to invoke the macro and Figure 3e is the drop-down box for switching between the play page and the replay page.

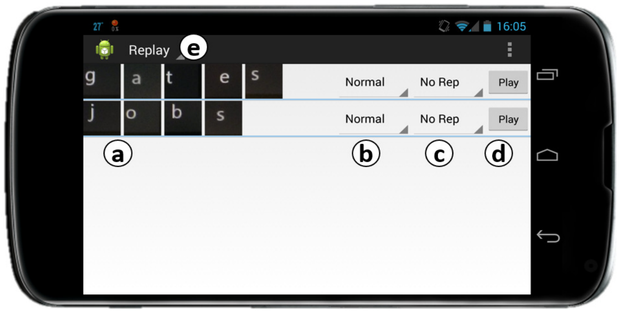


Figure 3. Replay page. This page contains the list of *play-macros*. The user can replay the macro by double tapping the item or by clicking the play button (d). Other UI elements include (a) the representative thumbnail of the macro. (b) the box for replay mode. (c) the box for replay timer. (e) the box for navigation between play page and replay page.

SCENARIOS

Pick-and-Play provides four *play-macro* replay modes: **normal replay**, **fast replay**, **slow replay**, **synchronous replay**. In the first three modes, *Pick-and-Play* replays the macro at different speed. Whereas the synchronous replay mode happens on the play page. When the SYNC button is toggled on (Figure 2), users' inputs are synchronously replayed on the remote device. In following paragraphs, we describe the usage of replay modes with practical scenarios.

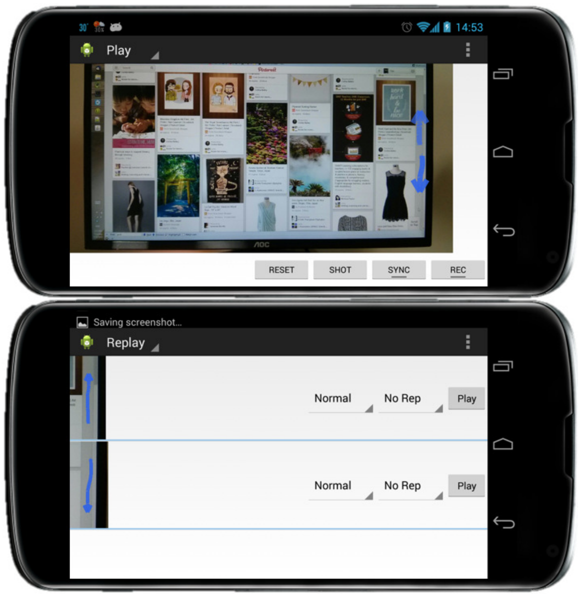


Figure 4. Normal Replay. The user first recorded browsing gestures, e.g. swiping up and down (top). Then with the *play-macros* on the replay page (bottom), he could control the remote device while stay in the favored postures and distances.

Normal Replay: Daily Interaction Made Easy

One major benefit of our action-based framework is its ability to transform daily interactions into reusable *play-macros*. Users can create their own interaction lexicons and build a personal macro dictionary for daily use.

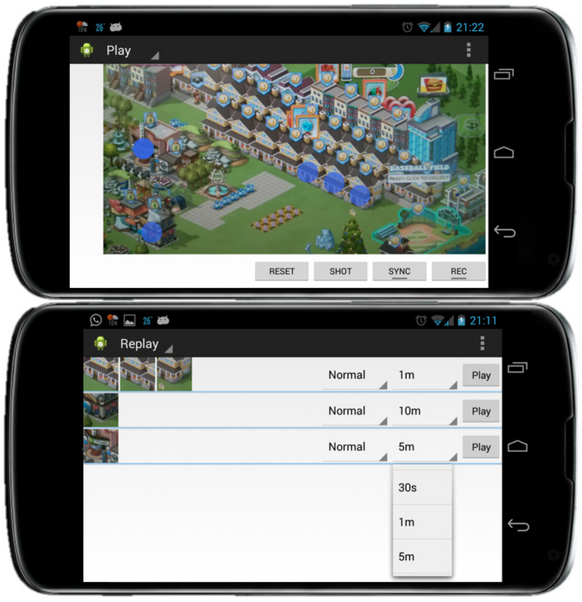


Figure 5. Periodic Replay. Social games usually require users to repeatedly clicking on the virtual objects. Users could create *play-macros* (top) with periodic timer.

For example, when browsing a long article on the desktop display, the user would have to repeatedly reach to the touch screen or the mouse for page scrolling. Such simple yet repeated input actions usually make the user unconsciously lean toward to the display or keep hands on the input devices on the desk. As a result, the user cannot stay in the comfortable postures, such as leaning on the chair. With *Pick-and-Play*, the user could first create the *play-macros* of dragging gestures (for web scrolling) or tapping gestures (for e-book page turning). Then she could easily access these controls through the mobile device while staying in the favored postures and distances.

Figure 5 shows another scenario of simplifying the repeating daily interaction with *Pick-and-Play*. Many web browser games require users to periodically click on specific spots of the screen. In this example, the user created macros for clicking on virtual objects. Through the *play-macro* list (Figure 5 bottom), the user could set up the repeating timer for each macro and let *Pick-and-Play* periodically play these macros.

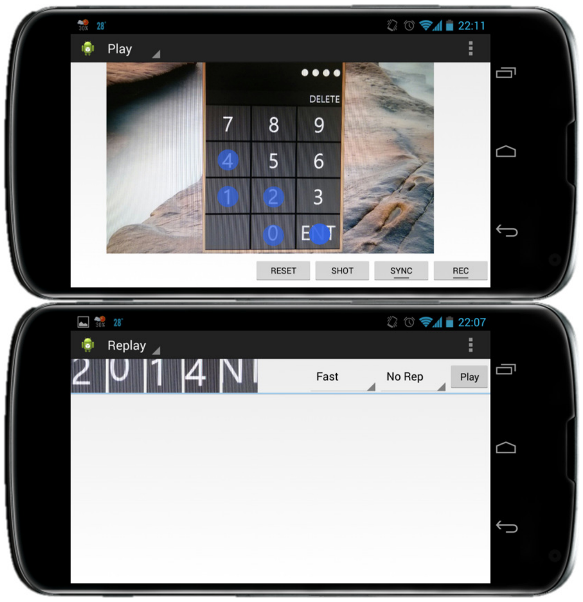


Figure 6. Fast Replay. With *Pick-and-Play* the user could bring the sensitive data input from public display onto personal device. In this pin code input example, the user input the pin code locally, then replay the macro in fast speed to prevent others from stealing the code.

Fast Replay: Sensitive Data Input in Public Space

Protecting the sensitive data in the environment with multiple-users or public display has received attention from researchers [15, 23] and is increasingly vital in our daily lives. For example, the user might have to enter the password or private chats through on-screen keyboard in front of others when playing video games with his friends in the living room. Alternatively, an engineer might have to unlock the pattern lock or enter the pin codes of a computer in front of guests. Even if we hide or encrypt the visual feedbacks, such as pin code numbers or motion patterns, the intentional attackers can still steal the passwords by observing the hand gestures or mouse movements [29].

With *Pick-and-Play*, users can bring sensitive inputs onto their personal mobile device. Figure 6 shows the scenario where the user *picked* the password input pad to her mobile device and input the password locally. Then, she replay the macro in the fast speed on the remote machine to avoid the potential shoulder surfing attacks.

Slow Replay: More Precise Control

Pick-and-Play also provides the slow replay mode, which is particular useful when users desire more precise control.

Figure 7 shows the scenario where the user was watching a video clip of a beautiful goal. However, he encountered two interaction problems: 1) the real goal clip was only about 50 seconds out of the 11:51 seconds long video and 2) there was no slow motion function. As a result, to review the goal, he had to repeatedly drag the video slider to the beginning of the goal clips, and emulate the slow motion play by moving the mouse cursor in a constant slow speed.

With *Pick-and-Play*, the user can build a *play-macro* by picking up the goal clips with drag gestures. Then the user can revisit the goal clip with one single click on the macro list and play the clip in slow speed by replaying the *play-macro* of video bar dragging in slow replay mode.

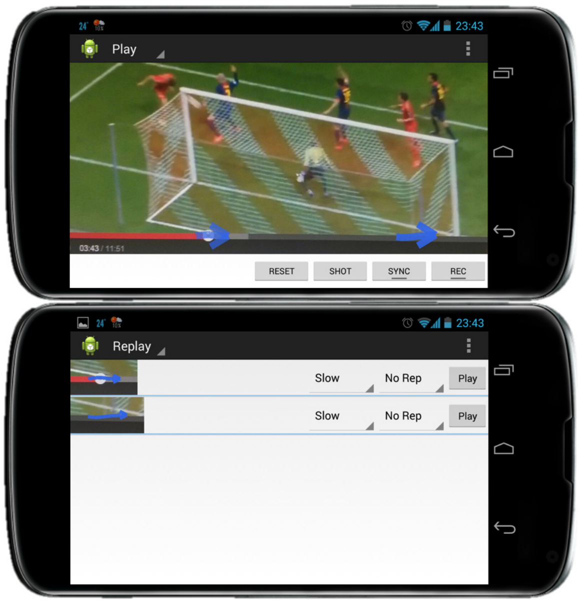


Figure 7. Slow Replay. With *Pick-and-Play*, the user could *pick* the favored clips from the long video and repeatedly replayed them in arbitrary speed.

Synchronized Play: Input Mapping

Unlike previous three asynchronous modes, in the synchronized play mode, *Pick-and-Play* does not translate inputs into *play-macros* but directly replay them on remote display after coordinate transformation. In this mode, the user stayed on the play page and performed inputs on the captured images, similar to the freeze mode in Touch Projector [6].

In this mode, *Pick-and-Play* complements the original input method on target device with the benefits of multi-touch inputs. It is especially beneficial in the scenarios when only sub-optimal input methods are available to the users or when the touch panel is in a distance.

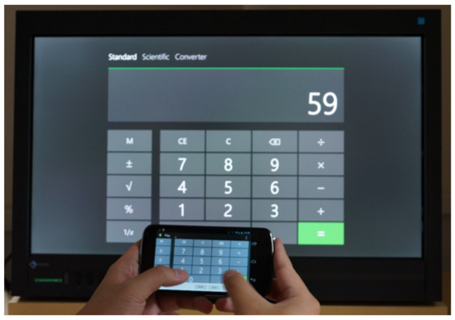


Figure 8. Synchronized Replay. *Pick-and-Play* can translate user inputs onto remote machine in real-time and bring the touch functions to non-interactive display.

Figure 8 demonstrates an example where the user *picked* the UI of calculator on a remote display and carried out the input tasks using the touch input on the smart phone.

ALGORITHM AND IMPLEMENTATION

At the core of *Pick-and-Play* are two main algorithms: **screen matching** and **input transformation**. The former establish the coordinate system mapping between the displays on the mobile device and remote device and the later translate the touch input sequences into mouse input sequences. In the end of the section, we also briefly describe the implementation on the remote machine side.

Screen Matching

Upon the initial connection of *Pick-and-Play*, the remote machine captures a screenshot, I_{remote}, and transfers it to user's mobile device. Afterward, whenever the user take the picture, I_{local}, the system automatically runs the homography calculation routine that calculates the coordinate mapping between I_{local} and I_{remote}. The *Pick-and-Play* deploys a lazy update approach on I_{remote} that it only requests I_{remote} when failing to establish the correct homography (the average re-projection error of feature points is larger than 10 pixels).

The *Pick-and-Play* utilize the standard image matching algorithm, similar to DeepShot [9]. The homography calculation routine first downsamples I_{local} and I_{remote} to half resolution for acceleration. It then extracts the feature points from both I_{local} and I_{remote} using Speeded-Up Robust Features (SURF) algorithm and calculates the best matching pairs with Fast Approximate Nearest Neighbor Search Library (FLANN). Finally, the homography between images is calculated using the RANSAC-based method (Figure 9).

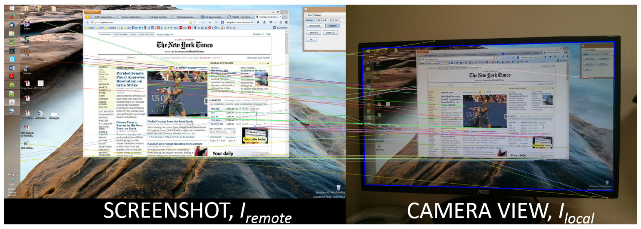


Figure 9. A screen matching example. Left the screen shot (I_{remote}) and right the image captured by phone camera (I_{local}). Colorful lines between images indicates feature points. The blue rectangle in camera view is constructed by connecting the four projected corners of I_{remote} on I_{local}.

Note that an alternative to capturing image with the phone camera is to directly requesting the screenshot from the remote device. *Pick-and-Play* provides multiple options to users, including always use captured photos, always request screenshots from remote machine, and request the screenshot when the matching algorithm fails.

The default option for our prototype is to always use captured photo. It is because the captured photo better fits users' perspective, the user can arbitrarily capture sub-regions of the screen, and this option consumes less bandwidth, due to the lazy updating policy. Nevertheless, our core ideas of the action-based framework and the *play-macro* are applicable for all options.

Input Transformation

With the homography between devices, we can map the inputs on local display onto remote one. However, two issues remain. First, the input method on local and remote device might differ, e.g. the touch input on mobile device and the mouse input on desktop computer. Second, the touch input is less precise due to the fingertip size and the occlusion by fingers.

In the desktop environment, commonly used mouse inputs are *clicking* and *dragging*. The Android device also provides an event listener for *press event*, *release event*, and *movement event*. However, unlike its desktop counterpart, the events are much noisier. In particular, when a user's finger is on the surface, the device would continually issue *press events*, and some accidental fingertip rolling could falsely trigger the movement event. As a result, a naïve direct mapping between the touch event and the mouse event could produce undesired or confusing results. The detailed investigation about the input point model is out of the scope of this paper. More discussions can be found in [4, 12, 28].

In the following paragraphs we describe the heuristic algorithms we used for categorizing touch input sequences into *clicking* or *dragging* and the algorithm for converting the touch input positions to mouse positions.

Categorization of Touch Inputs

First, we define a complete touch input sequence as a sequence that starts with a *press event* and ends with a *release event*. To categorize the sequence, we calculate the mean and standard deviation of the 2D coordinates in the input sequence. If the pixel distance of two standard variance is smaller than 320 pixel, then we categorize it as a *clicking* sequence, otherwise a *dragging* sequence. The number of 320 pixel corresponds to 26mm (1 inch) on our prototype device (Google Nexus 4, 320 dpi), which is the average size of adult thumb as well as the minimum targeting size reported in [8].

Coordinate Conversion from Touch Inputs to Mouse Inputs

The conversion between touch input and mouse input involve the precision problem. Researches showed that touch input increasing target time and error rate comparing to more precise input method such as stylus or mouse [12, 28]. Fortunately, nowadays most smart phone equip high mega pixel camera (8MP camera, 3264x2176 photo, for Google Nexus 4). It implies that after taking the photo with *Pick-and-Play*, users can still zoom-in the picture and compensate the imprecise touch input with larger UI element size.

In *Pick-and-Play* we deployed heuristic algorithms on the *clicking* and *dragging* sequence. For the clicking input sequence, we simply take the average of warped touch positions as the mouse clicking position. For the *dragging* sequence, we interpolate the potentially sparse touch input positions and obtain a sequence of dense mouse input positions for smoother replay quality. More specifically, we first warped every touch input points onto the screen space of remote display. Then for each consecutive point pair, we interpolate in-between pixel positions with the Bresenham's line algorithm [7]. In the end, we have a sequence of pixel positions, whose L1 distance is either one or two.

In early design phase, we considered using a b-spline approximation to smooth the line and remove the potential outliers, however, the computational cost is high and for the targeting scenarios described in the paper, current heuristic method already produced satisfying result.

Implementation on Remote Machine

For the remote machine, we implemented a light-weight application. It communicates with remote devices via sockets, captures screenshots and simulate the mouse input events via the Robot class in Java SDK.

TECHNICAL EVALUATION

We conducted a technical evaluation to test the feasibility of the screen matching algorithm and determine the proper working area of *Pick-and-Play*. Note that in the evaluation of the DeepShot [9], the independent variable was the tilting angle of the laptop display. We complement their evaluation by studying the feasible horizontal spanning angle for the image matching algorithm.

Environment Setup

In the evaluation, the remote device was a 13-inch Sony VAIO laptop with the screen resolution of 1920x1080. The mobile device was a Google Nexus 4 smart phone (4.7 inch display with 1280x768 resolution and an 8 MP camera capturing the image at 3264x2176 resolution). The evaluation was conducted in a room filled mostly with fluorescent light. Both the laptop and the phone were put on a flat table and initially the phone was 80 cm in front of the laptop display (point P in Figure 10a).

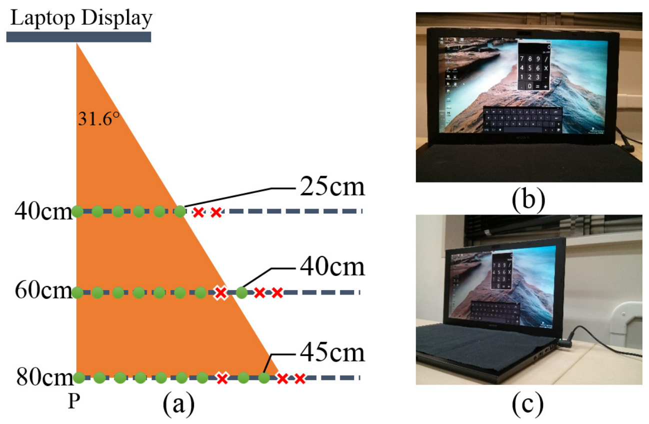


Figure 10. Technical evaluation. (a) shows the setup of the tech evaluation. Both the display and the phone are on the same flat table. Green dots indicates the successful image matching, red cross the failed ones. (b) is the photo captured from 40cm, 0°. (c) is the photo captured from 40cm, 36° (i.e. 30cm to the right), where the image matching algorithm failed.

Procedure

Starting from the point P in Figure 10, we moved the phone to the right by 5cm on the desk (along the dot lines in Figure 10a), pointing the phone camera to the screen, then captured a photo (e.g. Figure 10b). The procedure was repeated until the algorithm consecutively failed to estimate the correct homography twice (i.e. the average re-projection error of feature points is larger than 10 pixel). Figure 10c was a sample photo where our algorithm failed to estimate the correct homography. The same procedure was repeated at the distance of 40cm and 60cm to the laptop display.

Result and Discussion

In the 40cm case, the CV algorithm successfully estimated the correct homography until the phone is 25cm (32°) away from the center. In 60cm and 80 cm cases, the threshold distances are 40cm (33.7°) and 45cm (29.3°) respectively. We visualized these threshold distances as green dots in Figure 10. In sum, in our lab environment, the mean working viewing angle is about 63.2° (31.6° x 2) in front of the LCD.

As a post-experiment, we asked two participants to both randomly take 25 pictures in this 40 to 80 cm, 63.2° viewing angle area. Our screen matching algorithm

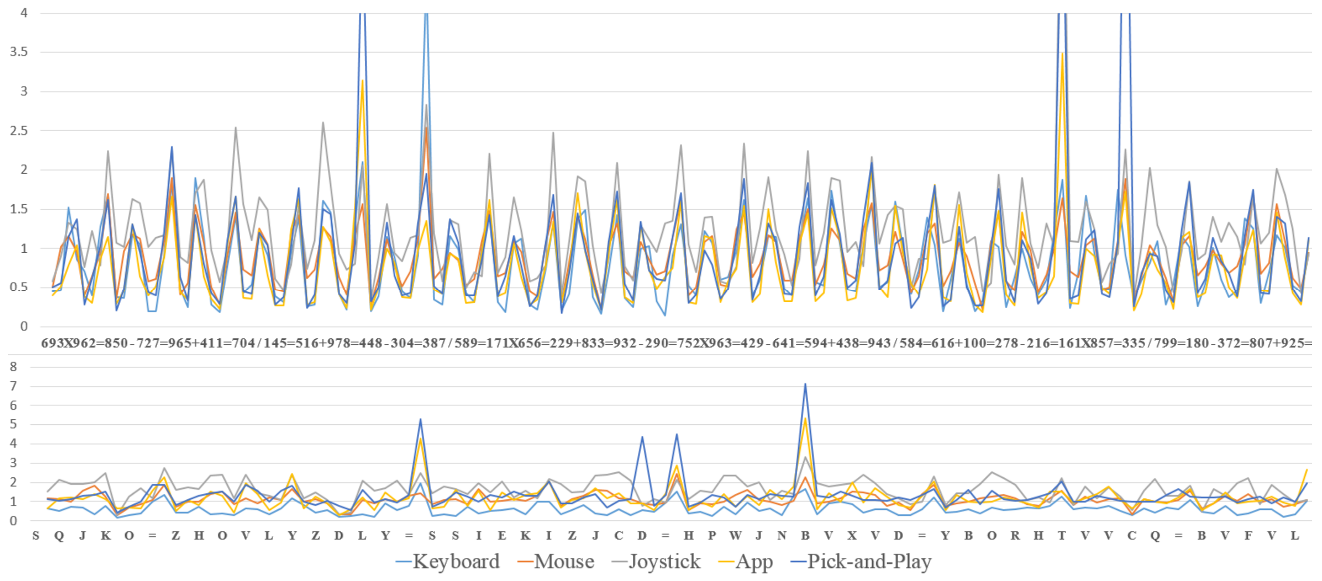


Figure 12. Per-char input time of equation (top) and text input task (bottom). The x-axis is the concatenation of input characters and y-axis the average per-character input time among users. Equal sign in x-axis equal to the end of input line.

performed robustly with the success rate of 86% (43 out of 50 attempts succeeded) in the test.

Still, the working range of *Pick-and-Play* is also largely related the quality of images captured by the phone camera, the size of the display as well as the computer vision algorithm for detecting and matching feature points. In our prototype, we did not use smart phone with the high-end camera nor did we extensively fine tune the screen matching algorithm. Nevertheless, this technical evaluation demonstrated that even with a mid-range mobile phone (Nexus 4), *Pick-and-Play* could still perform well in a reasonably wide working area.

QUANTITATIVE USABILITY STUDY

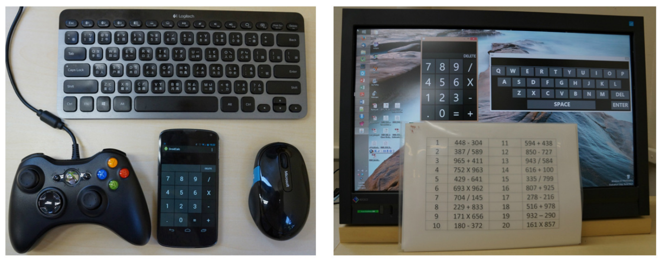
We described various scenarios where *Pick-and-Play* could help users achieve different tasks. However, the input performance of *Pick-and-Play* is not clear. In particular, how well can users interact with captured static images, in comparison to live user interfaces that provide real time visual feedback.

More specifically, *Pick-and-Play* requires users to capture a screenshot of the remote display, (optionally) translate or zoom the viewport and then perform inputs on the static image. When compared to a real user interface on the mobile device, *Pick-and-Play* introduces additional overhead of capturing and manipulating the photo and it requires users to interact with a static photo, which cannot provide visual feedbacks. For example, there are no appearance changes when the buttons are pressed and the resulting input texts only appear on the remote display.

Our hypothesis was that the lack of the local visual input feedbacks should not significantly reduce the input performance in the proposed scenarios. We examined the hypothesis through a quantitative user study that measured the input times and the error rates of five different input methods in two input tasks. Note that we did not expect *Pick-and-Play* to out-perform physical keyboard or the native input methods on the mobile device. However, if the hypothesis were supported by the study, we claimed that *Pick-and-Play* could be a reasonable alternative input method for the proposed scenarios.

Participants

Ten participants were recruited from our computer science institute. Four of them were native English speakers. All participants were used to the US keyboard layout and had been using smartphone with touch screen on daily-basis.



(a) Input Devices **(b) Task Setup**

Figure 11. User study setup.

Devices and Input Methods

Five input methods were used in the user study, including the keyboard, mouse, joystick, native mobile application and *Pick-and-Play* (Figure 11a). The keyboard was the one with standard 81-key US layout, the joystick was the xbox controller, and both the native mobile application and *Pick-and-Play* ran on a Google Nexus 4 phone (same as technical evaluation). Finally, an Eizo 24 inch screen with

1920x1080 resolution was used in the study. Note that in the test, *Pick-and-Play* was set to synchronized mode.

Tasks

The study consisted of two input tasks that were designed based on the scenarios described in previous sections.

Task 1: Equation Input

In the equation input task, every participant used five input methods to transcribe twenty equations, each consisting of two three-digit numbers and one operator. Among twenty equations, the number of appearance among digits and signs are equal.

Task 2: Text Input

In this task, every participant used five input methods to input ten password strings, each consisting of ten randomly picked English characters. This task used the random characters to minimize the carry over effects between input methods and to simulate the password input scenario.

The texts to be input were printed out and put into a transparent file folder (Figure 11b). The participants could place the folder at any favored position. There were five different sheets with different equation and text orders. All participants went through the sheets in fix order while the order of input methods were counter-balanced.

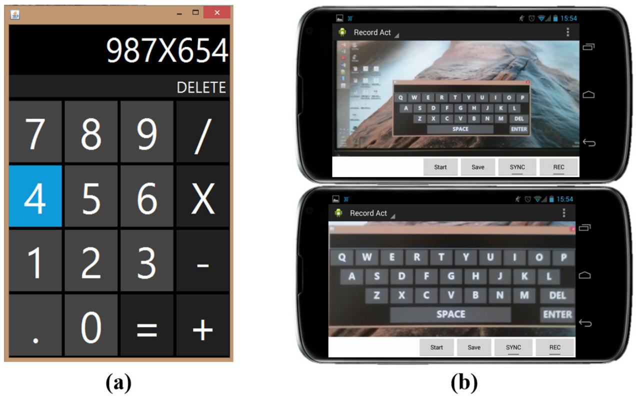


Figure 13. (a) On-screen calculator. Key 4 was selected and thus the different color. (b) Top: the preloaded photo in *Pick-and-Play*. Bottom: after the participants zoomed in the photo.

For both tasks, the participants were told to transcribe the scripts naturally. It was OK to have errors in the transcribed text and participants could fix errors if noticed, i.e. the *Recommended* error correction condition in [1].

Note that, the *Pick-and-Play* method preloaded a previously captured photo (top row of Figure 13b). Users started by zooming and translating the viewport (bottom row of Figure 13b), then performed the input tasks. We treated the photo capturing part as a control variable for better observing the correlation between the visual feedback and the input performance.

User Interfaces Feedbacks

Since the study was designed to evaluate how visual feedbacks affects the input performance, this section elaborates more on the user interface feedbacks of each input method.

When using keyboard, mouse and joystick, the input texts were shown on the remote display alongside the virtual calculator or keyboard (Figure 13). For the mouse and joystick methods, participants selected desired keys via the on-screen keyboard and calculator. The on-screen key changed its color when the cursor hovered over it or when it was the focus of joystick selection, e.g. Key4 in Figure 13a.

For the mobile application input method, the participant performed the tasks with the standard virtual keyboard and calculator in the Android OS. All feedbacks were turned on, including the vibration, sound and key pop-up. When using the *Pick-and-Play,* we only provided the vibration feedback upon the Android touch event.

Evaluation Result

Figure 14 shows the averaged input time of 10 participants for both equation and text input tasks. The red stacked bars on the *Pick-and-Play* column indicates the averaged zoom-in time, which are 4.23s for the equation input and 4.62s for the text input. The average width of the keys on the display were 15.1mm and 9.2mm respectively, which were both reasonable sizes for touch input [12].

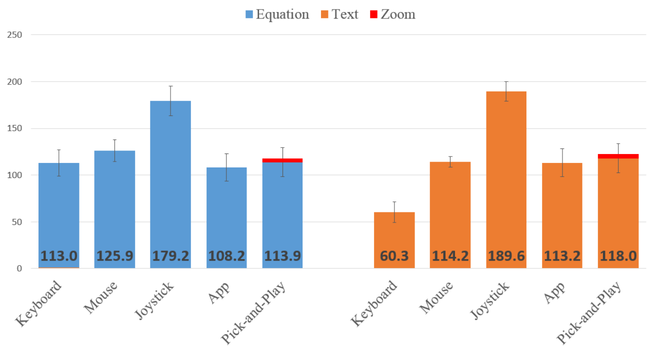


Figure 14. Average input time. Red stack bars show the averaged zoom-in time. Error bars indicates 95% CI.

For equation input task, the ANOVA test reports no significant difference among five input methods. For text input task, the ANOVA test reports significant difference (*F(1, 48)=7.07*, *p<0.05*) and the post-hoc t-tests reveals significant difference between the *Pick-and-Play* and all other input methods (all *p<0.05*) except Mouse. However, when excluding the zoom-in time, the post-hoc t-tests reports no significant difference between *Pick-and-Play* and the Mouse and *Pick-and-Play* and the App.

Figure 15 shows the input error rate for each input method with the *total error rate* metrics proposed by Soukoreff and MacKenzie [24]. The metrics has the advantage of being able to handle the natural input errors and corrections and thus fit our study well. For both tasks, error rates for all input methods are below 4%.

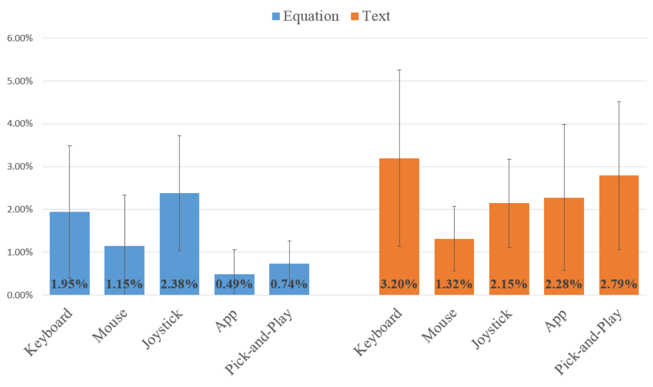


Figure 15. Total error rate [24]. Error bars indicates 95% CI.

Discussion

When excluding the initial zoom-in overhead, the evaluation result supports our hypothesis that the lack of local input feedbacks would not significantly affect the input performance. Note that in the real-world scenario, the initial overhead of the *Pick-and-Play* should also include the *pick* time, i.e. time to take a picture. However, since the overhead only happens in the beginning of the session, its impact should mitigate as the session lengthens. Also, after users constructing their *play-macros*, the subsequent uses of *play-macros* will not have this overhead cost.

We also observed similar input behaviors among users from the collected data. In particular, by concatenating all characters of the input tasks as x-axis and average per-char input time as y-axis, we have Figure 12. In the equation input task (top row), the small peaks of input times align fairly well with the positions of the first char of the three-digit number. In the text input task, despite less clear, small peaks can be found right after the equal sign, i.e. at the start of the new string.

Our observations suggests that these peaks happened as the participant moved his attentions among the input device, the typescript sheets, and the remote display. From the graph, this behavior is almost identical for all five input methods among all participants, regardless of the presence of local visual feedback or not.

It is also interesting to note those tallest peaks in the graph. The input log shows that these peaks corresponds to the point where the user were fixing previous incorrect inputs. The facts that these tallest peaks are from the *Pick-and-Play* could imply that the lack of local visual feedback and the less familiar input paradigm could have caused longer error fixing times. However, the low input error rate mitigates this potentially longer error fixing time in our user studies.

In sum, the evaluation result supports our hypothesis and suggest that the *Pick-and-Play* is a better or a reasonable alternative input method in various scenarios described earlier in the paper.

DISCUSSION AND FUTURE WORKS

The *Pick-and-Play* features unique asynchronous replay functions that could be useful in various scenarios. However, the asynchronous replay requires the position of user interface elements, e.g. windows frame, buttons, on the screen to stay the same as the captured photo. It is not a big constraint for modern full screen applications in Android/iOS/Win8, e.g. log-in/chatting window, video player and on-screen keyboards, whose UI elements mostly stay in fixed places. However, for the traditional desktop environment, where the application windows might change its position, *play-macros* could become invalid if UI positions differs. In such cases, users could either rebuild the *play-macro* or switch to synchronize mode for direct manipulation. As a future work, we plan to track the UI positions with computer vision algorithms [30] and update the macros accordingly.

Current *Pick-and-Play* prototype supports the operation transformation between the mouse input and the touch input, which covers a wide range of devices such as desktop PC, smart phone and smart TV. However, for some special closed systems that only support inputs from specific controllers, we plan to build a dedicated authoring UI helping users manually creating the mapping between input actions. We believe such UI will also encourage users to author personal input mappings, such as associating multi-touch gestures to specific functions [17].

Finally, our current prototype assumes an established connection between the mobile device and the remote device. We are currently investigating the possibility of pairing devices with the NFC or Bluetooth.

CONCLUSION

We propose *Pick-and-Play*, a light-weight asynchronous action-based framework. Complementing previous state-based frameworks, the *Pick-and-Play* migrates input actions and can directly work with existing applications. With the *Pick-and-Play*, users can easily transform daily interactions into reusable *play-macros*, which can be replayed in different modes for scenarios such as daily interaction simplification, sensitive data protection, and control precision enhancement. The technical evaluation shows that the *Pick-and-Play* has a reasonable working area and the quantitative user study suggest it a reasonable alternative input method in the targeting scenarios.

REFERENCES

1. Ahmed Sabbir Arif and Wolfgang Stuerzlinger. (2009). Analysis of text entry performance metrics, *Science and Technology for Humanity, IEEE Toronto International Conference*. 100-105
2. Rafael Ballagas, Michael Rohs, and Jennifer G. Sheridan. (2005). Sweep and point and shoot:

phonecam-based interactions for large public displays. *ACM CHI*. 1200-1203.

3. Dominikus Baur, Sebastian Boring, and Steven Feiner. (2012). Virtual projection: exploring optical projection as a metaphor for multi-device interaction. *ACM CHI*. 1693-1702.
4. Xiaojun Bi, Yang Li, and Shumin Zhai. (2013). FFitts law: modeling finger touch with fitts' law. *ACM CHI*. 1363-1372.
5. Sebastian Boring, Manuela Altendorfer, Gregor Broll, Otmar Hilliges, and Andreas Butz. (2007). Shoot & copy: phonecam-based information transfer from public displays onto mobile phones. *ACM Mobility*. 24-31.
6. Sebastian Boring, Dominikus Baur, Andreas Butz, Sean Gustafson, and Patrick Baudisch. (2010). Touch projector: mobile interaction through video. *ACM CHI*. 2287-2296.
7. Jack Elton Bresenham. (1965) Algorithm for computer control of digital plotter. *Journal of IBM System.* 4, 25.
8. Anthony Hall, James Cunningham, Richard Roache and Julie Cox. (1988). Factors affecting performance using touch entry systems: Tactual recognition fields and system accuracy. *Journal of Applied Psychology*, 4, 711–720.
9. Tsung-Hsiang Chang and Yang Li. (2011) Deep shot: a framework for migrating tasks across devices using mobile phone cameras. *ACM CHI*. 2163-2172.
10. David Dearman and Jeffery S. Pierce. (2008). It's on my other computer!: computing with multiple devices. *ACM CHI*. 767-776.
11. Sunao Hashimoto, Skihiko Ishida, Masahiko Inami and Takeo Igarashi. (2011). An Augmented Reality Based Remote Robot Manipulation. *ICAT*.
12. Christian Holz and Patrick Baudisch. (2011). Understanding touch. *ACM CHI*. 2501-2510.
13. Brad Johanson, Greg Hutchins, Terry Winograd, and Maureen Stone. (2002). PointRight: experience with flexible input redirection in interactive workspaces. *ACM UIST*, 227-234.
14. Shunichi Kasahara, Ryuma Niiyama, Valentin Heun, and Hiroshi Ishii. (2013). exTouch: spatially-aware embodied manipulation of actuated objects mediated by augmented reality. *ACM TEI*. 223-228.
15. David Kim, Paul Dunphy, Pam Briggs, Jonathan Hook, John W. Nicholson, James Nicholson, and Patrick Olivier. (2010). Multi-touch authentication on tabletops. *ACM CHI*. 1093-1102.
16. Yang Li and James A. Landay. (2008). Activity-based prototyping of ubicomp applications for long-lived, everyday human activities. *ACM CHI*. 1303-1312.
17. Hao Lü and Yang Li. (2012). Gesture coder: a tool for programming multi-touch gestures by demonstration. *ACM CHI*. 2875-2884
18. Robert C. Miller and Brad A. Myers. (1999). Synchronizing clipboards of multiple computers. *ACM UIST*. 65-66.
19. Nick Pears, Daniel G. Jackson, and Patrick Olivier. (2009). Smart Phone Interaction with Registered Displays. *IEEE Pervasive Computing* 8, 2, 14-21.
20. Jeffrey S. Pierce and Jeffrey Nichols. (2008). An infrastructure for extending applications' user experiences across multiple personal devices. *ACM UIST*. 101-110.
21. Jun Rekimoto. (1997) Pick-and-drop: a direct manipulation technique for multiple computer environments. *ACM UIST*, 31-39.
22. Daisuke Sakamoto, Koichiro Honda, Masahiko Inami, and Takeo Igarashi. (2009). Sketch and run: a stroke-based interface for home robots. *ACM CHI*. 197-200.
23. Dominik Schmidt, Julian Seifert, Enrico Rukzio, and Hans Gellersen. (2012). A cross-device interaction style for mobiles and surfaces. *ACM DIS*. 318-327.
24. William Soukoreff and Scott MacKenzie. (2003). Metrics for text entry research: an evaluation of MSD and KSPC, and a new unified error metric. *ACM CHI*. 113-120.
25. Andrew D. Wilson and Maneesh Agrawala. (2006). Text entry using a dual joystick game controller. *ACM CHI*. 475-478.
26. Desney S. Tan, Brian Meyers, and Mary Czerwinski. (2004). WinCuts: manipulating arbitrary window regions for more effective use of screen space. *ACM CHI*. 1525-1528.
27. Masayuki Tani, Kimiya Yamaashi, Koichiro Tanikoshi, Masayasu Futakawa, and Shinya Tanifuji. (1992). Object-oriented video: interaction with real-world objects through live video. *ACM CHI*. 593-598.
28. Daniel Vogel and Patrick Baudisch. (2007). Shift: a technique for operating pen-based interfaces using touch. *ACM CHI*. 657-666.
29. Keita Watanabe, Fumito Higuchi, Masahiko Inami, and Takeo Igarashi. (2012). CursorCamouflage: multiple dummy cursors as a defense against shoulder surfing. *SIGGRAPH Asia Emerging Technologies*. Article 6
30. Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. (2009). Sikuli: using GUI screenshots for search and automation. *ACM UIST*. 183-192.