GROUP ID: 5

GUI Interaction on Autopilot

Christina Chung
UTORid: chungc37
Teaching Labs ID: g4rice

Sahand Akbari
UTORid: akbaris6
Teaching Labs ID: akbaris6

Mohammad Kianpisheh
UTORid: kianpish
Teaching Labs ID: kianpish

ABSTRACT
By making objects tangible and directly manipulable, the graphical user interface provides an easy-to-use interface for non-programmers to operate a computer. But because these interfaces suffer from an inherent lack of programmability, users are often subjected to the tedium of manually repeating their tasks. We propose Autopilot, a context-aware system that detects and automates repetitive behavior. We validated Autopilot against the baseline of manual input, and show that not only does the system substantially reduce the amount of time it takes for participants to perform a repetitive task, but many expressed an openness toward incorporating the system into their practices.

INTRODUCTION
Thoughts, feelings, and beliefs are easily conveyed when humans communicate with one another. For instance, if a person were to point to a pile of papers and ask a friend to throw it away, the instruction would be well understood. This is due in part by peoples' gifted abilities to grasp the context of a situation [6]. Unfortunately, the same cannot be said when humans communicate with computers. What is considered to be a "pile" of papers? Which pile of papers is to be thrown away? Getting computers to understand such contextual information is a problem that has long stymied the development of rich user experiences. Only until recently, with advancements in artificial intelligence, have computers gained some ability to understand its user's intentions.

In Dey's conception, *context-awareness* is the ability of a computer "to provide relevant information and/or services to the user" [6]. While there are several forms of context-awareness, one is the automation of a task. In this work, we address the problem of automating repetitive tasks, particularly those performed on the graphical user interface (GUI). In essence, a repetitive task is a sequence of actions that is repeated multiple times in succession, such as renaming several files in a directory. Tasks such as these can be tedious for the able-bodied, and challenging for the impaired [7], making them well-suited for automation.

We propose Autopilot, a context-aware system that detects and automates repetitive behavior by leveraging the *programming by demonstration* (PBD) paradigm [4]. In PBD, users demonstrate the task to be automated, while the computer attempts to intuit the behavior. In sum, this work aims to enrich user experience by lessening the tedium of performing repetitive tasks.

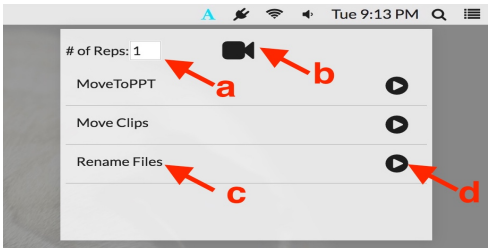

Figure 1. A screenshot of the Autopilot system. (a) Program the number of repetitions that the system should detect before prompting the user to execute a macro; (b) record a macro; (c) a macro (click to edit); (d) execute a macro.

RELATED WORK
By making objects tangible and directly manipulable, the GUI provides an easy-to-use interface for non-programmers to operate a computer [17]. However, these interfaces suffer from an inherent lack of programmability, subjecting users to manually repeating actions [14]. As a result, users have expressed a desire for automation [12].

One of the more largely attempted approaches to automate GUI interaction is through PBD, in which computers learn how to perform tasks from user-demonstrated behavior [4].

For instance, SMARTedit automates the process of formatting text in a text editor based on a few examples provided by the user [10]. CoScriptor enables users to save previously performed actions in web-processes for future reuse [11]. In these works, the tasks that are automated are bound to single applications, and do not support tasks involving several.

One primary challenge in supporting automation on a cross-application level is knowing where GUI elements are positioned, as this information often not readily available. To recognize GUI elements, a popular technique is to leverage computer vision, using various forms of template matching and feature matching [13,18]. These techniques have been used by a number of applications: providing contextual help when interacting with a desktop computer [19], testing GUI elements [2], context-aware video tutorials [3,16], and GUI task automation [7]. Yet, computer vision techniques break down when GUI elements substantially change in appearance. As the purpose of Autopilot is to propose better interaction techniques rather than to make advancements in GUI element detection, Autopilot will be implemented in a mock operating system to avoid the hassle of dealing with current technological limitations.

GROUP ID: 5

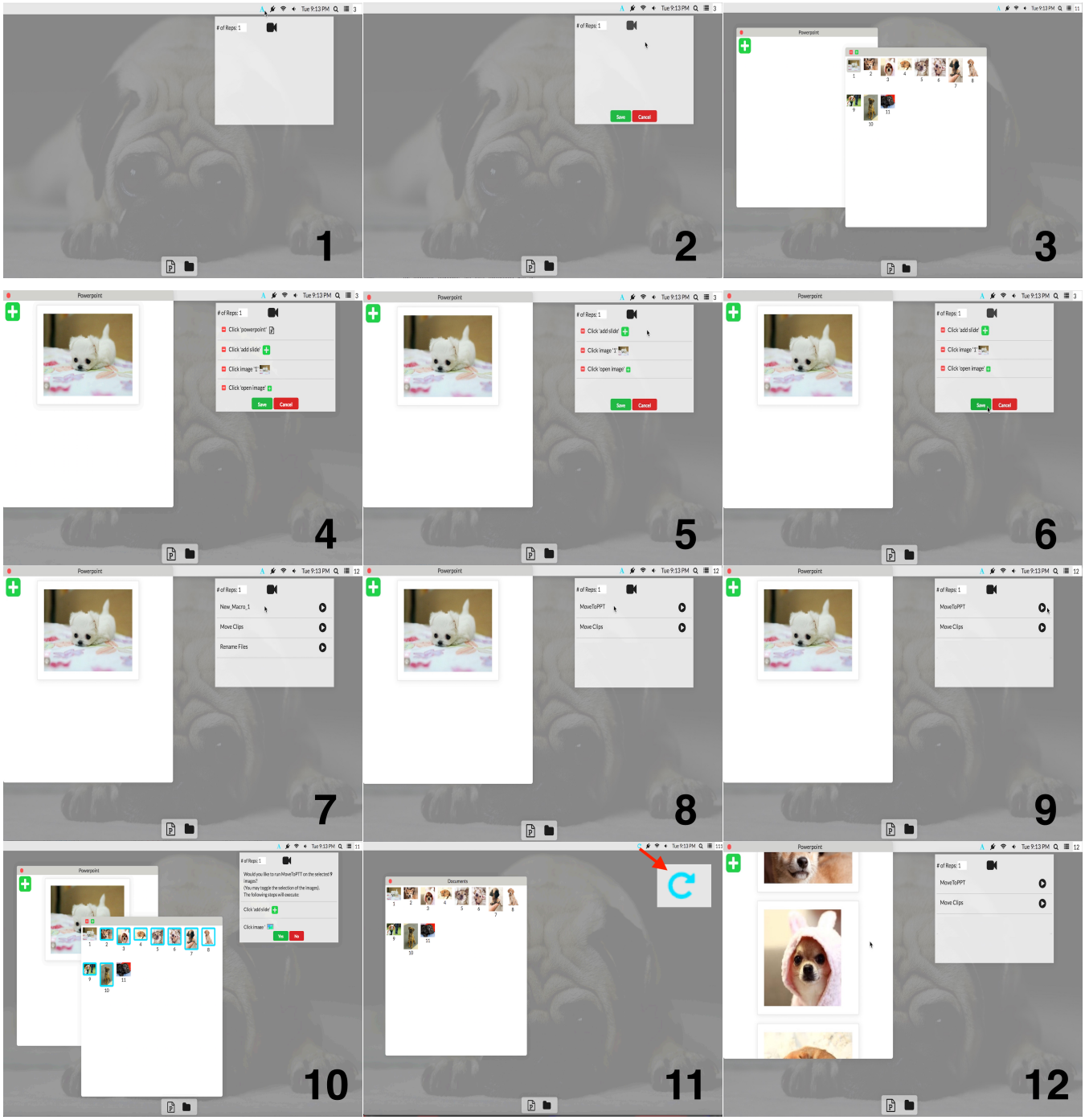

Figure 2. An example of importing images into PowerPoint with Autopilot. 1-7) generating the macro; 8) editing macros; 10-12) executing a macro on demand. (Please refer to "Example Usage Scenarios" for a detailed description of this storyboard.)

The design of Autopilot was shaped by two works in the literature, Help, It Looks Confusing (HILC) [7], which supports automation of GUI interactions on desktop computers, and SUGILITE, which accomplishes a similar goal for mobile devices [8].

At present, identifying repetitive actions without prior knowledge is a complex problem that necessitates robust noise and sequence detection in high-volume, high-dimensional data [5]. To account for these setbacks, HILC and SUGLITE instead use macros to identify repetitive behavior, by requiring its users to pre-record actions that they would like to automate. Additionally, the macros produced by the two systems can generalize to new behaviors. For instance, one might teach SUGILITE to order a Starbucks Cappuccino but the macro for that task could be generalized to ordering an Iced Cappuccino as

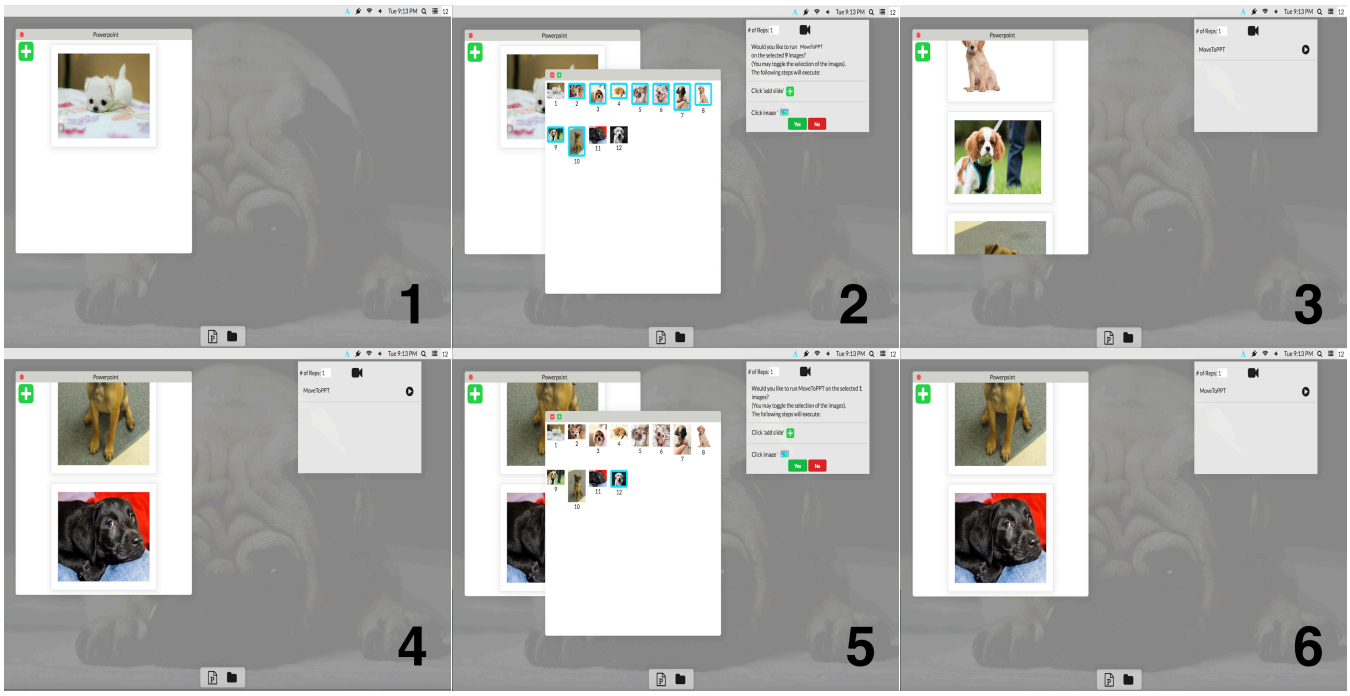

Figure 3. Example scenario of Autopilot detecting repetitive behavior (please refer to "Detecting Repetitive Behavior" for a detailed description of this storyboard.)

well. Our design takes a similar approach by producing macros from recorded behavior and generalizing these macros to other tasks.

Both systems come with its share of drawbacks. Firstly, SUGILITE and HILC's functionality for editing incorrectly learned macros could be improved. For example, users of SUGILITE have access to each macro's source code, for which they can modify to its intended behavior. In its evaluation, users did not perceive this feature to be useful, as the code was often difficult to parse, especially for those with limited programming experience. Similarly, HILC queries users with follow-up questions if an action is deemed ambiguous. This may diminish the user's sense of control, as they cannot modify the macros directly to their intentions, but must rely on the system to propose them [1]. A final shortcoming found in both systems is that, when tasks are automated, users literally see them being carried out on the screen. We believe this may be hindrance, as it precludes users from engaging in other activities on their computer while macros are executing. Updating the GUI repeatedly also consumes computational resources, leading to an increased processing time.

EXAMPLE USAGE SCENARIO

This section describes a situation where Autopilot may be useful: importing a many images into PowerPoint. This scenario is depicted as a storyboard in Figure 2 and Figure 3.

Creating Macros

Sarah would like to automate the process of importing ten images in her "Documents" directory into PowerPoint. To do so, she must first create a macro on the Autopilot application. She clicks on the Autopilot application on her computer's toolbar and the Autopilot application window appears (Figure 2 (1)). Then, she clicks on the "record" button to record the desired behavior (Figure 2 (2)). During the recording procedure, her GUI interactions will be logged by Autopilot. Each GUI interaction appears as a step within the Autopilot application window. Sarah now launches the PowerPoint application (Figure 2 (3)). Next, she clicks the "+" button, selects "1.png", and adds it PowerPoint. The image is then displayed within the slide deck (Figure 2 (4)). She has now completed all actions that she would like to automate. She now removes an unwanted step from the list actions logged by Autopilot: launching PowerPoint (Figure 2 (5)). Next, she clicks the "Save" button to save the macro (Figure 2 (6)). Autopilot then adds the macro to the top of the list of macros and is given the default name "New_Macro_1" (Figure 2 (7)).

Editing Macros

Sarah can edit or delete macros have been previously created, by clicking on their names. Sarah renames her newly created macro to be "MoveToPPT" and decides to delete "MoveClips" as well (Figure 2 (8)).

GROUP ID: 5

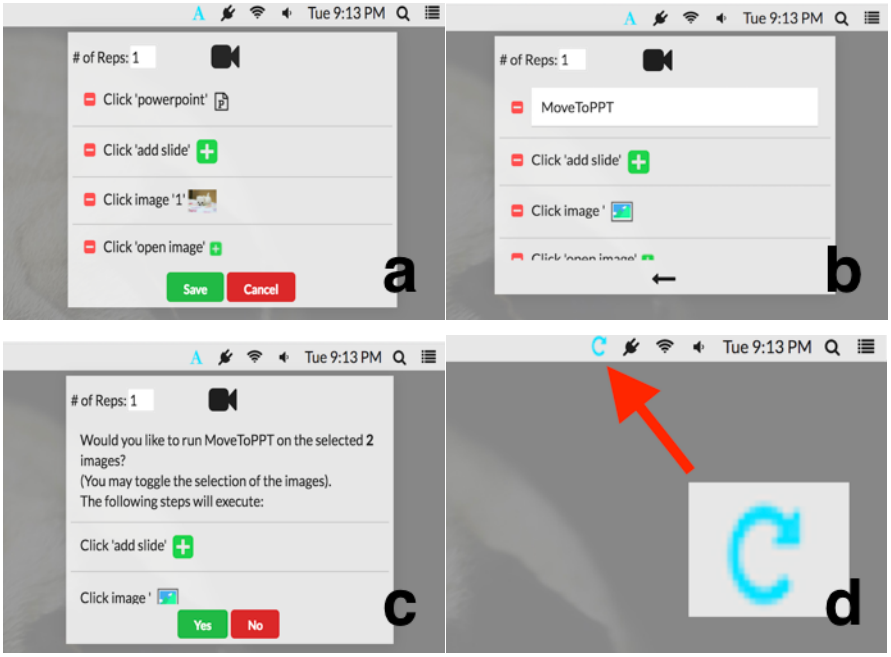

Figure 4. Four screenshots of Autopilot. (a) Recorded steps of a macro; (b) interface for renaming and editing macros; (c) prompter for executing a macro; (d) bottom right: execution of a macro.

Automating Macros

To transfer the rest of the images into PowerPoint, Sarah clicks the "MoveToPPT" macro's "Play" button (Figure 2 (9)).

Autopilot then prompts Sarah to confirm whether she would like to transfer the selected images into PowerPoint ("2.png" through "11.png"). Since she only wants up to "10.png" to be transferred, she deselects "11.png" and proceeds to click the "Yes" button to execute the macro (Figure 2 (10)). During the automation of a task, a circular circular progress bar will appear next the Autopilot icon on Sarah's desktop toolbar. This allows Sarah to know the that macro is under execution while attending to other activities on her computer. Once the task has completed, the progress bar disappears (Figure 2 (11)). The images are then transferred over to her slide deck (Figure 2 (12)).

Detecting Repetitive Actions

At a new point in time, Sarah would like to move the first ten images within her "Documents" directory into another PowerPoint project. Having forgotten that she created a macro to automate this task, she proceeds to add "1.png" into PowerPoint (Figure 3 (1)). Since she has programmed the number of repetitions to be detected to be once before the system recommends a macro to be run (Figure 1 (a)), when she proceeds to select "2.png", Autopilot predicts that a task is being repeated. The Autopilot window pops up and prompts Sarah to confirm whether she would like to transfer the "3.png through 12.png" into PowerPoint (Figure 3 (2)). Since she only wants up to "10.png" to be transferred into PowerPoint, she deselects "11.png" and "12.png". After doing so, she clicks the "Yes" button, and these images are then transferred over to her slide deck (Figure 3 (3)).

Sarah changes her mind. She now wants to include "11.png" in her slide deck, so she manually imports it into PowerPoint (Figure 2 (4)). Next, she inadvertently selects "12.png". Autopilot detects this as a potential repetitive behavior and displays a prompt to confirm whether she would like to transfer "12.png" into PowerPoint (Figure 3 (5)). Since she does not intend for this behavior, so she clicks "No" to cancel the prompt. Upon doing so, Autopilot does nothing and no longer recommends automating tasks until another macro gets repeated above the programmed threshold (Figure 3 6)).

THE Autopilot SYSTEM

The Autopilot system was designed to incorporate the five design goals that follow. Screenshots of Autopilot are shown in Figure 1 and Figure 4.

A Sense of Control

According to Barkhuus and Dey, context-aware applications can be characterized in two ways [1]. On the one hand, the system can carry out actions automatically regardless of the user's intentions. This is known as *active awareness*. *Passive awareness*, on the other hand, is when the system has contextual information, but does not act upon it without the user's permission. In their study, Barkhuus and Dey found that awareness compromises one's sense of control, since activities that one would normally carry out are now taken over by the system. However, participants reported preferring systems providing awareness over those that did not, as they were

accepting of less control if it was as a means to accomplishing a valuable goal.

Presently, learning by demonstration is still a challenge for computer scientists [9]. This is because unlike other approaches in machine learning, which have access to large amounts of training data, PBD systems must learn behavior from a few examples. A fully automatic Autopilot may spur unwanted consequences when its behavior counters users' desires. As a result, Autopilot was designed to be passively aware—it identifies whether users may be performing a repetitive task, but does not proceed to automate tasks without their confirmation (Figure 4 (c)).

Automate on Demand

Users have the option to execute a macro on demand (Figure 1 (d)), rather than having to perform the repeated actions based on the programming number of times (Figure 1 (a)) to get a prompt from Autopilot.

Directly Manipulable Interface

Autopilot incorporates a direct manipulation interface (Figure 1) [17]. While this seemingly contradicts its goals of alleviating the burdens of interacting with GUI, we felt that users would only be occasionally recording or modifying their macros. Most of their interactions would entail responding to Autopilot's recommendations for automating tasks. As a result, users may forget how to execute specific commands on Autopilot after an extended period non-use. A direct manipulation interface can mitigate this as the interfaces are intuitive to use [17].

An additional consideration that we had was providing keyboard shortcuts to enable users to respond to Autopilot's prompts, but this would add complications as the system would have to figure out whether a user's keyboard event was intended for Autopilot or for some other application.

Proximity-Occlusion Trade-off

Autopilot is shown as a dropdown window on the desktop computer's toolbar (Figure 1). We also played with idea of implementing a tooltip that would appear next to the user's cursor to reduce the distance that the user's cursor must travel when interacting with the system. However, we decided against it since the tooltip may occlude the user's view and result in user frustration.

Error Correction

In line with Intharah's suggestion that systems should "fail gracefully" [7], Autopilot has a corrective measure: users may remove any action that they do not want as part of a macro when recording (Figure 4 (a)) or anytime after (Figure 4 (c)). In Autopilot, each step of a macro is displayed with an accompanying screenshot and a descriptive text of the associated action. This was inspired by Myer's work on *program visualization*, which abstracts code with a visual representation, providing an easier way for people to understand and modify commands [15].

IMPLEMENTATION

The source code is available online (Appendix). The system was implemented in a web application emulating an operating system, and was developed using JavaScript (Angular JS and Node.js), HTML and CSS. In a real operating system, knowing where GUI elements are located is a challenge, and there is currently no way to perform cross-application GUI interactions without inducing mouse and keyboard events. Since we intend to evaluate whether user perceive the experience to be better when they do not have to literally see the GUI interactions being carried out, we will use a mock operating system to achieve this goal. Moreover, a mock-up that we've personally implemented will enable us to test and refine our designs more efficiently as we will have more familiarity with its inner workings.

As the system was primarily developed for demonstration purposes rather than for use in practice, events that were logged and monitored were limited to user clicks. The system currently supports only the following repetitive behaviours: adding images to PowerPoint, opening the PowerPoint app, and the operating system's file system explorer. Ideally, a real implementation of Autopilot would detect and automate all repetitive GUI interactions.

After a new event is logged, Autopilot will analyze the log to identify whether the user has engaged in repetitive behavior. We operationalize repetitive behavior as being a macro that has been repeated above a threshold that the user may specify (Figure 1 (a)). If Autopilot detects that one of the macros has been repeated above that threshold, Autopilot will recommend automating the task the next time the user performs its initial step.

EVALUATION

To evaluate Autopilot, we conducted a user study that compared the system to the baseline approach of manual input.

Participants

Eight participants (2f, 6m) were recruited by word-of-mouth. Given that HILC is of greatest semblance to our work and recruited seven participants for its evaluation [7], we followed to their footsteps in recruiting eight (i.e., seven, with one additional participant to counterbalance Autopilot with the baseline). Participants were aged 18-34 (M=25, SD=5.32). All participants were adept programmers, as measured on a scale from 1-5 (where 1=no experience, 5=very experienced): 3 (n=2), 4 (n=1), 5 (n=5). Participation in the study was voluntary and compensation was not provided.

Task and Apparatus

The experiment was carried out on a 13'' Macbook Pro with an Intel Core i5 processor (Figure 5 (a)). Participants performed GUI interactions using a Microsoft Wireless Mobile Mouse 4000 (Figure 5 (b)).

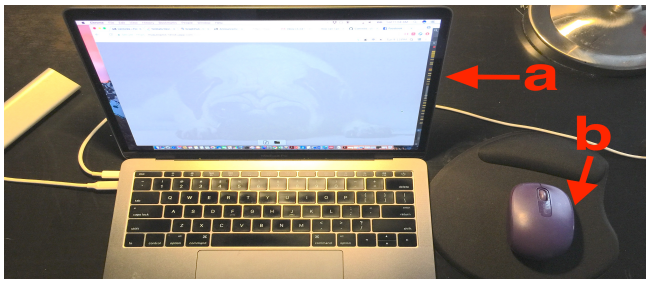

Figure 5. The experimental apparatus. (a) 13" MacBook Pro; (b) Microsoft Wireless Mobile Mouse 4000.

The task involved participants transferring images from a directory into PowerPoint, both manually (i.e., one image at a time, without any aids) and using Autopilot. For the study, the repetition threshold was set to one for consistency across participants.

Stimulus

Two factors formed the independent variables in this study: *interaction method* and *image count*. To compare Autopilot to the baseline of manual input, the interaction method factor was comprised of the levels: *manual*—in which the user manually performs the task of importing images into PowerPoint and secondly, *automated*—in which the user uses Autopilot to automate the task. The image count factor modulated the number of images that participants had to import into PowerPoint namely, three, six, nine and twelve images. These levels were realized because prior work has shown that three to six repetitions is the point at which automation breaks even with manual input, in regard to the task completion time. Yet, because it requires at least two images for Autopilot to recommend that a macro should be execute, our lowest level was three rather than one [12].

All factors were fully-crossed for each participant, providing $2 \times 4 = 8$ stimulus conditions.

Measures

A two-factor within-subjects design was employed. To mitigate ordering effects, the interaction method levels were counterbalanced, and the image count levels were randomized. That is, each participant began the study with one interaction method (either manual or automated), performed the tasks for each of the four image count conditions in a randomized order, and then proceeded to the second interaction method.

Our measures were both objective and subjective. In line with other work [7,12], the task completion time for each stimulus condition was recorded as a measure of performance. We considered the task completion time to be the elapsed time between the start of a stimulus condition to the time at which participants had successfully imported all specified images into PowerPoint. The subjective measures consisted of responses to a post-study questionnaire that participants filled out at the end of the experiment. The post-study questionnaires included ratings of the system's usefulness and usability on a 7-point Likert (Table 2), and an open-ended question regarding each participant's overall impression of Autopilot.

Procedure

Participants first signed a consent form, received information about the study task, and was provided with a walkthrough of the emulated operating system and Autopilot. Afterwards, participants then began the study—uploading images into PowerPoint for each of the stimulus conditions. Participants filled out the post-study questionnaire after completing all stimulus conditions.

RESULTS

On average, participants spent 15.37 (*SD*=2.48) minutes in the study.

Task Completion Time

Average task completions times for each of the stimulus conditions are shown on Table 1. To test for significance effects, a two-way repeated measures ANOVA was used. For brevity, only significant results are reported. The sphericity assumption was met for all reported effects.

There was a significant main effect of interaction method on the task completion time $[F(1,7) = 8.124,\ p =< .05, \eta^2 = .537]$. Image count also had a significant main effect on the task completion time $[F(3,21) = 20.654,\ p < .001,\ \eta^2 = .747]$. Further, the ANOVA test revealed a significant two-way interaction effect between image count and interaction method $[F(3,21) = 10.364, p < .001,\ \eta^2 = .597]$.

Pairwise cross-factor comparisons revealed a significant main effect of image count for the manual interaction method $[F(3,21) = 21.841, p < .001,\ \eta^2 = .757]$ (Figure 6). In this case, Tukey Bonferroni-corrected post-hoc tests indicated that the differences were between image counts three and six ($p < .005$), three and twelve ($p < .01$), six and nine ($p < .005$), six and twelve ($p < .005$), and nine and twelve ($p < .05$). There was also a significant effect of interaction method on an image count of nine [$F(1, 7) = 6.147$, $p < .05$, $\eta^2 = .468$] (Figure 7 left) and interaction method on an image count of twelve [$F(1, 7) = 20.717$, $p < .001$, $\eta^2 = .757$] (Figure 8 right).

In sum, these results suggest that when users must perform many repetitive actions, Autopilot substantially reduces the task completion time in comparison to the manual input approach.

Image Count	Manual	Automated
3	17590.88 ms	16316.00 ms
6	24946.88 ms	23685.25 ms
9	37574.25 ms	21772.12 ms
12	43813.50 ms	20885.12 ms

Table 1. Average task completion times in milliseconds for each stimulus condition.

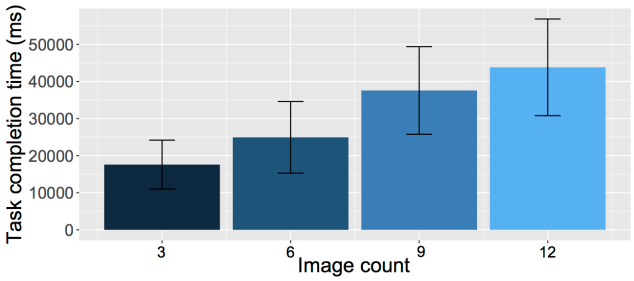

Figure 6. The mean and 95% confidence interval plots of the task completion times in milliseconds for each level of image count on the manual interaction method.

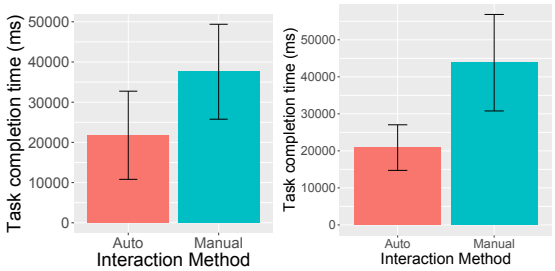

Figure 7. The mean and 95% confidence interval plots of the task completion times in milliseconds for each interaction method when the image count was nine (left) and when the image count was twelve (right).

Questionnaire Item	Mean
1. It's easy to learn how to use this system.	5.25
2. My interaction with the system is clear and understandable.	3.25
3. I'm satisfied with my experience using this system.	4.63
4. I find the system useful in helping me creating automation.	5.88
5. I find automating tasks with the system is efficient.	5.88
6. I would use this system to automate my tasks.	4.25

Table 2. Post-study questionnaire items and the mean responses as measured by a 7-point Likert scale (where 1=strongly disagree, 7=strongly agree).

Subjective Feedback

The Likert scale ratings for the post-study questionnaire items are summarized in Table 2. Ratings pertaining to the usability of Autopilot (Table 2 questions 1-3) were relatively mediocre. Generally, participants reported that the system was useful (Table 2 questions 4-6).

In the long answer responses, all participants expressed that Autopilot could be a useful tool for automation "With this tool, I could insert pictures faster especially when I had to put 12 pictures." (P5). However, participants pointed to improvements that could be made in several respects. For one, some participants (3/8) noted it was difficult to learn how the system "Not very intuitive at the beginning but after few repetitions I was able to understand more about the system." (P3)

It also seemed that the system's approach of informing the user about a macro's beheaviour, i.e., by displaying the sequence of steps that it will execute (Figure 4 (c)), was not informative, as some participants (2/8) were apprehensive about running a macro as they unsure of what it might do to the state of their operation system. For instance, P5 brought up that "It was totally unclear how my actions were recorded, and what was the 'recipe' that the system automatically created…It would be great if the system shows a preview so that I can expect what will happen". P7 underscored a similar concern "It could feel rather mysterious trying to use a macro if you haven't done the same one before—you don't know if it could go wrong!"

Finally, some participants expressed a desire for additional features that could integrated into the system, such as enabling users to specify the contexts and conditions for which a macro should execute "It would be cool if we can specify the context of condition" (P6), as well as programming the system to detect repetitive behaviour without the use of a macro "I had to click the record button…if the system recognized my behavior automatically, it would be much convenient and fast." (P8)

DISCUSSION

The evaluation of Autopilot was promising. Not only did participants perform tasks faster using the system, but many had expressed an openness toward incorporating the system it into their practices.

In spite of this, all participants thought that the system needed to be improved prior to making it available for widespread use, particularly by providing a more intuitive interface. Additionally, participants expressed concerns about not knowing how certain macro might alter state of their operating system. Autopilot could be made better by leaving no surprises, perhaps as suggested by P5, this can be achieved by showing a preview of a macro's behaviour. Further, the actions should be revertible if something goes awry.

Participants also seemed to want the system to accomplish more than its current capabilities. For instance, by defining the specific contexts for which automation should occur, and to detect repetitive behaviour purely from recognizing routine behaviour instead of employing macros.

LIMITATIONS AND FUTURE WORK

The evaluation of Autopilot is largely limited by the fact that all participants had prior experience in programming. As developing code is analogous to creating a macro ("rules than can be reused"), these participants would likely be more adept at using the system than those with no exposure to programming. The study's results are therefore prone to bias and future evaluations of Autopilot should aim for a more diverse group of users.

Additionally, future studies should examine how users might leverage the system in practice, rather than in a lab setting where the usage scenarios might feel artificial.

To progress GUI automation technologies, future work should be directed toward providing these systems with an

enhanced ability to understand its user intentions, thus making "macro-less" automators that can detect repetitive behaviour a palpable reality. This will entail improving recognition of human routine behaviours, and of the contexts and conditions for which particular repetitive tasks should be automated.

CONCLUSION

This work presents an account on the development and evaluation of Autopilot, a context-aware system that detects and automates repetitive GUI interactions, using programming-by-demonstration. The results of the study have shown that Autopilot substantially reduces the amount of time that users spend carrying out repetitive tasks, and the feedback from participants showed promise in its future adoption.

In sum, this work is stepping stone toward alleviating the tedium of repetitive GUI interactions, and a broader goal of advancing context-aware designs.

APPENDIX

Source code: https://github.com/chrchung/autopilot.

REFERENCES

1. Louise Barkhuus and Anind Dey. 2003. Is Context-Aware Computing Taking Control away from the User? Three Levels of Interactivity Examined. . Springer, Berlin, Heidelberg, 149–156. https://doi.org/10.1007/978-3-540-39653-6_12
2. Tsung-Hsiang Chang, Tom Yeh, and Robert C. Miller. 2010. GUI Testing Using Computer Vision. In *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*, 1535. https://doi.org/10.1145/1753326.1753555
3. Kai-Yin Cheng, Sheng-Jie Luo, Bing-Yu Chen, and Hao-Hua Chu. 2009. SmartPlayer: user-centric video fast-forwarding. In *Proceedings of the 27th international conference on Human factors in computing systems - CHI 09*, 789. https://doi.org/10.1145/1518701.1518823
4. Allen. Cypher, Daniel Conrad. Halbert, David Kurlander, Henry Lieberman, David Maulsby, Brad A. Myers, and Alan Turransky. 1993. *Watch what I do : programming by demonstration*. MIT Press. Retrieved November 14, 2017 from https://dl.acm.org/citation.cfm?id=168080
5. Himel Dev and Zhicheng Liu. 2017. Identifying Frequent User Tasks from Application Logs. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces - IUI '17*, 263–273. https://doi.org/10.1145/3025171.3025184
6. Anind K. Dey and Anind K. 2001. Understanding and Using Context. *Personal and Ubiquitous Computing* 5, 1: 4–7. https://doi.org/10.1007/s007790170019
7. Thanapong Intharah, Daniyar Turmukhambetov, and Gabriel J. Brostow. 2017. Help, It Looks Confusing: GUI Task Automation Through Demonstration and Follow-up Questions. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces - IUI '17*, 233–243. https://doi.org/10.1145/3025171.3025176
8. Toby Jia, -Jun Li, Amos Azaria, and Brad A Myers. SUGILITE: Creating Multimodal Smartphone Automation by Demonstration. https://doi.org/10.1145/3025453.3025483
9. Tessa Lau. 2009. Why Programming-By-Demonstration Systems Fail: Lessons Learned for Usable AI. *AI Magazine* 30, 4: 65–67. https://doi.org/10.1609/aimag.v30i4.2262
10. Tessa Lau, Steven A. Wolfman, Pedro Domingos, and Daniel S. Weld. 2003. Programming by Demonstration Using Version Space Algebra. *Machine Learning* 53, 1/2: 111–156. https://doi.org/10.1023/A:1025671410623
11. Gilly Leshed, Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. CoScripter: Automating & Sharing How-To Knowledge in the Enterprise. Retrieved November 15, 2017 from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.120.8647
12. Toby Jia-Jun Li, Amos Azaria, and Brad A. Myers. 2017. SUGILITE: Creating Multimodal Smartphone Automation by Demonstration. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems - CHI '17*, 6038–6049. https://doi.org/10.1145/3025453.3025483
13. D.G. Lowe. 1999. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1150–1157 vol.2. https://doi.org/10.1109/ICCV.1999.790410
14. B.A. Myers. 1992. Demonstrational interfaces: A step beyond direct manipulation. *Computer* 25, 8: 61–73. https://doi.org/10.1109/2.153286
15. B. A. Myers, B. A., Myers, and B. A. 1986. Visual programming, programming by example, and program visualization: a taxonomy. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '86*, 59–66. https://doi.org/10.1145/22627.22349
16. Suporn Pongnumkul, Mira Dontcheva, Wilmot Li, Jue Wang, Lubomir Bourdev, Shai Avidan, and Michael F. Cohen. 2011. Pause-and-play: Automatically Linking Screencast Video Tutorials with Applications. In *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*, 135–144. https://doi.org/10.1145/2047196.2047213
17. Shneiderman. 1983. Direct Manipulation: A Step Beyond Programming Languages. *Computer* 16, 8: 57–69. https://doi.org/10.1109/MC.1983.1654471

GROUP ID: 5

18. 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*, 183–192. https://doi.org/10.1145/1622176.1622213
19. Tom Yeh, Tsung-Hsiang Chang, Bo Xie, Greg Walsh, Ivan Watkins, Krist Wongsuphasawat, Man Huang, Larry S. Davis, and Benjamin B. Bederson. 2011. Creating Contextual Help for GUIs Using Screenshots. In *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11*, 145. https://doi.org/10.1145/2047196.2047214