2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)
October 11-14, 2020. Toronto, Canada

Repurposing Visual Input Modalities for Blind Users: A Case Study of Word Processors

Hae-Na Lee[1], Vikas Ashok[2], I.V. Ramakrishnan[1]

[1]Department of Computer Science
Stony Brook University
Stony Brook, NY, USA
{haenalee, ram}@cs.stonybrook.edu

[2]Department of Computer Science
Old Dominion University
Norfolk, VA, USA
vganjigu@odu.edu

***Abstract*—Visual 'point-and-click' interaction artifacts such as mouse and touchpad are tangible input modalities, which are essential for sighted users to conveniently interact with computer applications. In contrast, blind users are unable to leverage these visual input modalities and are thus limited while interacting with computers using a sequentially narrating screen-reader assistive technology that is coupled to keyboards. As a consequence, blind users generally require significantly more time and effort to do even simple application tasks (e.g., applying a style to text in a word processor) using only keyboard, compared to their sighted peers who can effortlessly accomplish the same tasks using a point-and-click mouse.**

This paper explores the idea of *repurposing* visual input modalities for non-visual interaction so that blind users too can draw the benefits of simple and efficient access from these modalities. Specifically, with word processing applications as the representative case study, we designed and developed *NVMouse* as a concrete manifestation of this repurposing idea, in which the spatially distributed word-processor controls are mapped to a virtual hierarchical 'Feature Menu' that is easily traversable non-visually using simple scroll and click input actions. Furthermore, NVMouse enhances the efficiency of accessing frequently-used application commands by leveraging a data-driven prediction model that can determine what commands the user will most likely access next, given the current 'local' screen-reader context in the document. A user study with 14 blind participants comparing keyboard-based screen readers with NVMouse, showed that the latter significantly reduced both the task-completion times and user effort (i.e., number of user actions) for different word-processing activities.

***Index Terms*—Accessibility, assistive technology, screen reader, visual impairment, word processing**

I. Introduction

Blind users rely on special-purpose assistive technology, namely a screen reader (e.g., JAWS [1], VoiceOver [2], NVDA [3]), to interact with computer applications. A screen reader narrates the screen content serially, and with the aid of this narration, a blind user navigates and accesses different parts or components of the application using keyboard shortcuts. However, most applications manifest an intricate and dense arrangement of different components and features in their GUIs that are by design, more suitable for a visual 'point-and-click' interaction with a basic pointing device such as a mouse, as opposed to a serial shortcut-based keyboard interaction. Therefore, blind screen-reader users find it arduous and tedious to do even simple activities (e.g., access ribbon commands) in applications by relying only on the keyboard.

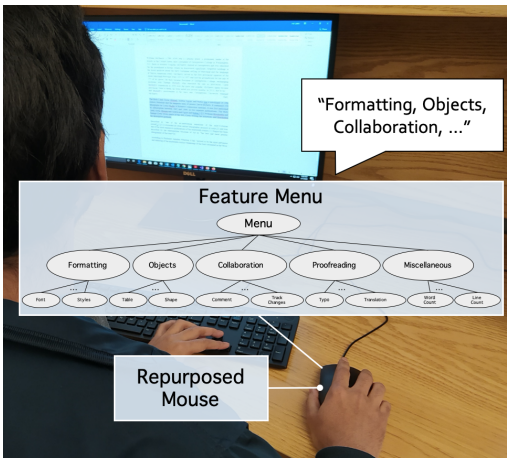

Fig. 1. A blind user accessing application features using NVMouse's repurposed mouse instead of keyboard. The user (i) does a middle click to access the custom Feature Menu, (ii) scrolls to find the desired feature in the menu, (iii) does a left click to access controls associated with the feature, (iv) scrolls to find the desired control, and (v) left-clicks on the desired control.

To address the usability-related shortcomings associated with keyboard-only screen-reader interaction, in this paper, we explore the idea of transforming or "repurposing" the existing visual input modalities intended for sighted users into convenient-to-use non-visual input modalities for blind users. In this regard, we selected word processing applications as the vehicle for our investigation, as they exemplify the typical GUIs of computer applications that consist of a main work area with several auxiliary commands and features spatially distributed around this work area. Furthermore, the word processing applications are commonly used in everyday lives of blind users, and proficiency with the applications has been recognized as an important skill for the employment of blind individuals [4], [5].

To understand the scope and magnitude of the usability problems faced by blind screen-reader users with word processing applications using just the keyboard, we conducted a user study with 10 blind users. The study revealed that the fundamental bottleneck impeding users' productivity was the

tedious and frustrating process of accessing the application features corresponding to different user activities. Furthermore, using only the keyboard for all activities such as accessing formatting commands, typing, navigating document content, reviewing content changes, adding and resolving comments, etc., caused a lot of disorientation, shortcut mix-ups, and other unintended errors due to repeated context switching.

Informed by the findings of the preliminary study, we designed and developed *NVMouse*, a 'scroll-and-click' input interface designed by repurposing (reprogramming) a computer mouse, which serves as an auxiliary non-visual input modality for accessing important application features, in addition to the keyboard. For example, as shown in Figure 1, using the simple scroll-and-click actions available in the mouse, blind users can quickly and easily access the various ribbon commands (e.g., `Font`, `Styles`, etc.) via a custom *Feature Menu*, without *losing any context*. Similarly, the blind user can also access the list of comments in the `Collaboration` feature group in the menu, and scroll over them one-by-one. Thus, with NVMouse, blind users can reap the benefits of the computer mouse in word-processing activities, akin to their sighted peers.

To further enhance efficiency, for a select category of frequently-used ribbon commands such as formatting, NVMouse leverages a custom-trained data-driven prediction model to dynamically reorder these commands under the `Formatting` feature group in the Feature Menu, based on their likelihood of being used next, given the user's current screen-reader context in the document. The reordering places the commands most likely to be used next at the front (i.e., first child) in the group, making them "within easy reach" for blind users – akin to "point-and-click".

II. Related Work

Our work closely relates to extant research on alternative non-visual input modalities and usability of word processors.

A. Alternative Input Interfaces for Blind Users

To compensate for the lack of non-visual alternatives to convenient-to-use visual input modalities, alternative input modalities for blind users have been previously explored [6], [7]. For example, the multimodal audio-haptic interface proposed by Doush et al. [6] enables screen-reader users to access the content of Excel charts. However, their interface only helps blind users *consume* existing content easily, but does not support creating or editing it, as well as accessing other application features. In the IBM Home Page Reader (HPR) [7], the numeric keypad was used as an auxiliary input interface for navigating web pages. Besides the need to remember numerous shortcuts, HPR only supported information consumption in webpages, which has vastly different user-interaction needs compared to that of general productivity applications, such as Word supported by NVMouse.

Analogous to HPR's numeric pad, Apple's MacBook Touch Bar [8] provides contextual menus and navigation shortcuts for a variety of productivity applications. But these suggestions are manually engineered, and the design of Touch Bar is primarily for visual consumption, thereby requiring screen-reader users to spend significant time exploring and orienting themselves each time they want to access the suggestions on it. Similar to the Touch Bar, Apple's built-in screen reader, VoiceOver, also provides access to commands via its rotor feature. However, these commands mainly assist in navigating content. Speed-Dial [9], like VoiceOver's rotor, also supports easy hierarchical navigation of content via its external Microsoft Surface Dial input interface. Khurana et al. [10], on the other hand, propose spatially interaction techniques that leverage the keyboard surface to easily facilitate non-visual interaction with 2D structures. However, both these works exclusively focus on non-visual web browsing and are limited to content navigation, which in essence is tantamount to reading the web page elements. In contrast, interaction with applications such as word processors not only involves navigating content, but also frequently accessing the auxiliary spatially-distributed application content, such as command ribbons, menus, sidebars, comments and version history, etc.

B. Non-visual Usability of Word Processors

Despite the importance of word processing applications, there is a dearth of studies on improving the usability of these applications for blind users [11]–[15]. Moreover, almost all of these works primarily focus on improving the interaction experience for only a specific aspect of word processors. For example, Mori et al. [11] only focused on accessibility, where they made Google Docs more accessible by providing virtual overlays, without considering usability or user-interaction strategies. Morales et al. [14], on the other hand, focused only on formatting; they propose guidelines for a support tool in Microsoft Word that can assist blind users format their documents independently. Recent works have focused on accessibility and usability of collaborative features in word processors [15], [16]. For example, an accessible prototype for collaborative writing was suggested [15]. The prototype enables blind users to exploit the right-click context menu to access all the comments and document revisions, and also accept/reject these revisions. Waqar et al. [16] also focused on the accessibility of collaborative features of word processors, by providing audio notifications to blind users informing about changes made by other collaborators.

In contrast to the aforementioned research, we propose a single cohesive interface framework that enables blind users to easily perform assorted word-processing related activities such as document navigation, formatting, reviewing, proofreading, collaborating, etc. Besides we augment the framework with a novel prediction model that provides nearly "instant access" to an important category of application commands. Moreover, unlike the existing works, our approach is informed by the blind users' interaction behavior and interaction strategies with word-processing applications, which we gathered via a preliminary user study with blind participants.

III. Uncovering Usability Issues

To understand the usability issues that blind users face while interacting with productivity applications using only keyboard, and also obtain insights regarding how to "repurpose" a mouse for convenient non-visual interaction, we conducted a user study with 10 fully blind word-processor users.

A. Participants

The 10 participants were recruited through local mailing lists and word-of-mouth. The average age of participants was 43.8 (Median = 41, SD = 10.8, Range = 30-61), and the gender representation was equal (5 male, 5 female). Initial screening was done via phone interviews to enforce the initial criteria requiring participants to be frequent users (or experts) of JAWS and Microsoft Word. All participants also stated that JAWS was their preferred screen reader. Many participants (especially elderly) did not own laptops, but had desktops. Even participants owning laptops stated that they used desktops at school/work.

B. Apparatus

The study was conducted using an Acer laptop running Windows 10, with JAWS 16 and Word 2016 installed. A traditional external keyboard was plugged into the laptop.

C. Design and Procedure

The participants were asked to create a document (in Microsoft Word) with a title, a heading, and a bulleted list with two items. They were then asked to track and accept changes, add three comments, and navigate document content and comments while checking for grammar and spelling mistakes. All information such as the textual content and formatting style (e.g., font, font size, and font color) for each task was given, and a concurrent think-aloud protocol was adopted.

Before the tasks, the participants were given practice time (~10 minutes) to familiarize with the keyboard, and customize JAWS and other computer settings to suit their preferences. All user actions were captured using a screen recording software, and a keystroke logger was used to record individual keystrokes. All conversations during the study were in English.

D. Notable Findings

Repeated switching of screen-reader context. All participants repeatedly switched the screen-reader focus back-and-forth between the main edit-area and the other application features concerned with formatting, review, proofreading, and collaboration. This was due to the fact that there was only one cursor (i.e., keyboard) that the participants could leverage while interacting with the application.

Problem in navigating grids. Five participants struggled to locate the desired formatting command options for commands such as `Font Color` and `Text Styles` that appeared in a 2D grid, despite JAWS reading out instructions on how to navigate the grid. These participants at first used only the *UP* and *DOWN* arrow keys to loop through the first column of the grid before realizing that additional options could be accessed with the *LEFT* and *RIGHT* arrow keys.

Excessive key presses for accessing formatting commands. To find and apply most formatting commands (e.g., font, styles, color, etc.), almost all (eight) participants repeatedly pressed the *TAB* hotkey to serially navigate through numerous formatting commands in the application ribbon, before finding the desired command. These users stated that they cannot remember hundreds of shortcuts associated with different application features, and therefore rely only on a few basic navigational ones that let them surf through the features one-by-one. Only two participants who knew the complex Word shortcut (*CTRL+SHIFT+P*) for opening a separate format dialog box could do most of the formatting commands in one iteration by setting the different parameters. Nonetheless, they too accessed the ribbon to apply the commands not available in the format dialog box.

Re-orientation after every non-edit activity. Eight participants, in at least two instances, pressed arrow keys after performing a non-edit activity (e.g., adding a comment, applying a font, etc.), to reorient themselves within the main edit-area. They stated that they did this because they had forgotten what they had typed and where the cursor was, prior to shifting screen-reader focus away from the main edit-area to access other application features. However, four participants, in at least one instance, did not realize that the focus had shifted back to the edit-area after completing an activity, and therefore unintentionally modified the existing content by pressing the shortcut keys in the edit-area.

E. Summary

The study observations indicate that almost all of the usability problems stem from the fact that blind users predominantly rely only on the keyboard to do both typing and other activities, whereas sighted users can efficiently split the interaction effort between the keyboard and mouse, i.e., using the keyboard for editing and the mouse for quickly accessing other application features via point-and-click. One way to address these usability problems is through a "separation of concerns" by using the keyboard only for editing, and a non-visual "repurposed mouse" for accessing other application features. In this regard, we propose the NVMouse assistive technology described next.

IV. NVMouse Design

The fundamental design goal of NVMouse is to provide a streamlined access to various application controls associated with different word-processing activities for blind users; the present spatially-distributed 2D layout of application controls is not favorable for non-visual interaction [10].

Figure 2 presents an architectural schematic of NVMouse. With NVMouse, the computer mouse is adapted or 'repurposed' such that the blind user can interact with a word processor using both the keyboard and the mouse modalities; presumably the user can utilize the keyboard for typing and

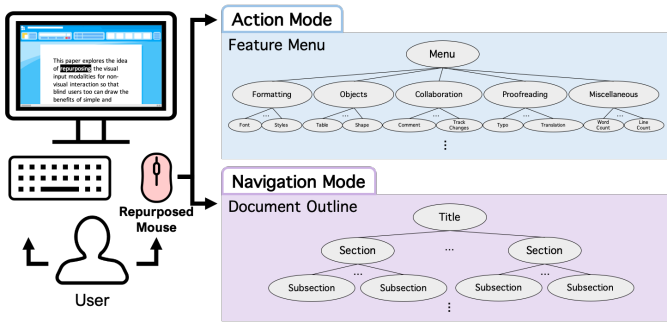

Fig. 2. An architectural schematic of NVMouse.

editing content, and the mouse for performing authoring activities, such as formatting, reviewing changes, adding/resolving comments, navigating content, etc.

NVMouse operates in two modes (Figure 2): *navigation mode* and *action mode*. By default, NVMouse is in navigation mode, i.e, mouse actions are associated with navigating the document content structure hierarchically (e.g., title, sections, subsections). Action mode, on the other hand, is triggered by a middle click. In this mode, the mouse input gestures are reprogrammed to facilitate user interaction with a hierarchically-organized custom *Feature Menu* that provides alternative access to the assorted application features. Also, the controls or commands in the frequently-accessed *Formatting* feature group of the Feature Menu are dynamically reordered based on their likelihood of being used next, given the current screen-reader context in the document.

A. Custom Feature Menu

The Feature Menu consists of feature groups (see Table I), where each feature group is associated with one of the word-processing activities, e.g., formatting, proofreading, etc. Each feature group contains all associated application controls arranged in the form of a tree, with an abstract node containing the name of the feature (e.g., "Collaboration") as the root. Table I details the list of feature groups that we have manually-defined in NVMouse, along with their constituent application controls. We selected these features and associated controls after interviewing 20 proficient blind users who indicated what activities they frequently performed in word processors, and what application controls they wished to be easily accessible. As also mentioned in Section II, these features have also been individually explored by other works [6], [13]–[15].

Note that some controls (e.g., `Insert Row`) are activated only when applicable. The first group (child) of the Feature Menu is *Formatting*, as it is the most-frequent user activity in word processing. Also, unlike other groups where the order of application controls is manually fixed, the order of controls (i.e., children) in the Formatting group is dynamically determined by the custom prediction model (Section IV-C).

B. Redefining Mouse Actions

1) Accessing Feature Menu: Table II lists the mouse actions and their corresponding functions as redefined by NVMouse for interacting with the Feature Menu. The *middle click* toggles the NVMouse mode between navigation and action. By default, scrolling moves the focus between the feature groups when the menu is first activated. A *left click* selects a feature group, and the subsequent scroll actions will then move the focus between the top-level controls in the menu tree. Once the desired control is found, another *left click* will either activate it if it doesn't have sub-controls (e.g., `Bold`) or move the focus down the menu tree to the first sub-control (e.g., font color options) of that control (e.g., `Font Color`). To close the currently focused sub-control and move focus up the tree to the parent control, the user needs to execute a *right click*. Regardless of the focus, a *middle click* deactivates the Feature Menu and toggles the mode back to navigation.

TABLE I
FEATURE GROUPS AND THEIR FEATURE CONTROLS.

Feature Group	Top-level Application Controls
Formatting	Font, Font Size, Font Color, Alignment, Bold, Underline, Italic, Line Spacing, Bullet List, Fill Color, Styles, Borders
Objects	Insert [Table, Picture, Shape, Symbol, Equation, Row, Column], Delete Row, Delete Column, Merge Cells
Collaboration	Insert Comment, View Comments, Next Comment, Previous Comment, Delete Comment, Track Changes, View Changes, Accept All Changes, Next Change, Previous Change, Accept Change, Reject Change
Proofreading	Next Typo, Previous Typo, All Typos, Next Error, Previous Error, All Errors, Dictionary, Translation
Miscellaneous	Word Count, Character Count, Line Count, Current Page Number, Margins, Insert Page Break

2) Document Navigation: Table II also lists the mouse actions and their corresponding functions as redefined by NVMouse for navigating document content. NVMouse treats the entire document content as a sequence of paragraphs (Open XML format[1]), each having a specific outline level (an integer between 0 and 9); lower outline levels represent nodes higher up in the outline tree (e.g., outline level 1 for title, section headings), whereas larger outline levels indicate nodes lower in the tree (e.g., outline level 3 for subsubsection headings). By leveraging this "*paragraph outline*" information in the document metadata, NVMouse constructs a document outline tree. This outline tree is then coupled with the mouse interface as defined in Table II. Note that navigating to a node in the outline tree corresponds to moving the screen-reader focus to the associated portion of the document content.

To keep track of updates to the document structure, NVMouse monitors all user edits. If a user's edit results in alteration of the overall document outline, NVMouse rebuilds the outline tree. We found the overhead of recomputing the tree to be negligible (less than 500 ms for 200+ page documents).

[1]http://officeopenxml.com/

TABLE II
MOUSE ACTIONS AND THEIR RE-DEFINED FUNCTIONS IN TWO MODES.

Action Mode	
Mouse Action	**Function**
Middle click	Close the Feature Menu
Scroll	Scroll through feature groups, or controls at the same tree level
Right click	Go up one level in the menu tree
Left click	Activate a control or go one level down in the menu tree
Navigation Mode	
Mouse Action	**Function**
Middle click	Open the Feature Menu
Scroll	Scroll through the siblings or elements at the present depth in the document tree
Right click	Go up one level in the document tree
Left click	Go down one level in the document tree

C. Prediction Model for Formatting

The ordering of commands in the Formatting group is dictated by a custom prediction model. We implemented the prediction model in the form of a multi-class classifier, with each of the commands representing a class. The class scores from this classifier were then used to determine the command order in the Formatting group, with the command with the highest score placed first. We trained the classifier using example documents scraped from the web, as explained next.

1) Command Dataset: We first scraped $6,000$ Word document templates and examples from the web. This collection comprised diverse document types, such as CV, statement, report, letter, thesis, etc. Since these documents were complete with all the formatting already applied throughout their content, we used them to generate ground truth dataset. Specifically, for each command that we identified using the Office Word Primary Interop Assembly service in a document, we extracted several features representing the *local context* in which that command was applied, and then created an example $(\mathbf{x}, c)$ with the command c and the context feature vector $\mathbf{x}$. The dataset comprised a total of $2,728,962$ command examples. To learn a model, this dataset was randomly split into three parts (60% for training, 20% for validation, and 20% for test sets).

In our dataset, we observed that the most commonly-used commands were: *Styles*, *Alignment*, *Line Spacing*, *Format* (e.g., *Bold*, *Italic*), *Font*, *Font Size*, *Font Color*, *Bullet List*, and *Insert* (e.g., *Insert Picture*). Since the occurrence of other commands were negligible, we built our prediction model for only these nine types of formatting commands. Interestingly, the existing context menu in Microsoft Word application too contains mostly these nine types of commands, although most of them are *not accessible* with a screen reader.

2) Attribute Representation: We hand-crafted command attributes to fit the word processing application domain. We defined the local context of an applied command to be its containing paragraph, based on our observation in the collected documents that most of the application commands are applied at the granularity of either paragraph (e.g., `Alignment` command) or selection within a paragraph (e.g., `Bold` command). Therefore, the attributes can be categorized into two groups: (i) *paragraph-level* – e.g., total word count, style, alignment, `Line Spacing` command usage; and (ii) *selection-level* – e.g., selection word count, selection position (within the paragraph), `Font` command usage, `Font Size` command usage, `Font Color` command usage, usage count of other commands on a selection, etc. For each command c, the embeddings of these two groups of attributes were concatenated to generate the corresponding feature vector $\mathbf{x}$. The categorical attributes (e.g., *alignment* feature) were represented using one-hot encoding.

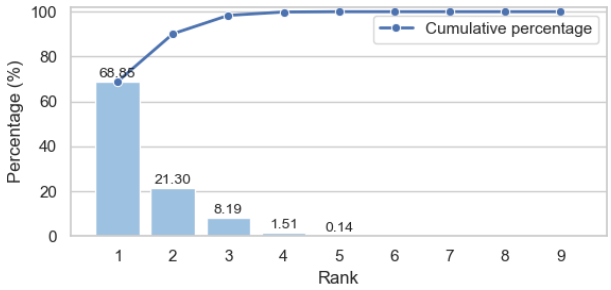

Fig. 3. A rank histogram computed on the test set and cumulative percentage as the rank increases.

3) Model Architecture: We trained a neural network model on our dataset with three hidden layers having 120, 84, and 25 units, respectively. A linear transformation was applied in each layer, and rectified linear unit (ReLU) was used as an activation function. To predict the likelihood of commands being used next, we tapped into the scores of all the commands given by the last hidden layer.

4) Model Evaluation: We assessed the performance of prediction model using mean reciprocal rank (MRR) metric, as we are interested in the rank of only one command, i.e, class, in the model output, and not all the commands. MRR is defined as the average of the inverse of the ranks for a set of command examples C:

$$MRR = \frac{1}{|C|}\sum_{i=1}^{|C|}\frac{1}{r_i}$$

Here r_i is the rank of the command c_i in the predicted command list. Overall, our prediction model yielded a high MRR value of 0.8264.

Figure 3 shows both the distribution of ranks predicted by our model for the command examples in the test set, as well as the cumulative percentage of commands over the rank dimension. As seen in Figure 3, 68.85% of commands were correctly ranked first, thereby enabling the users to instantly access the command in the Feature Menu. Also, 99.85% of commands were ranked within the top 4, which indicates that with the prediction model, almost all desired commands can be accessed with at most three mouse scrolls.

5) Comparison with the Default Context Menu: As mentioned earlier, Microsoft Word supports a default context menu listing several commonly-used commands. The menu can be brought up by right-clicking a mouse, and it is also accessible by a menu key in keyboard. However, unlike our menu with prediction model, the order of commands within the context menu is fixed. For the same aforementioned nine frequently-used commands, the fixed command order in the context menu was as follows: *Font*, *Font Size*, *Styles*, *Format*, *Font Color*, *List*, *Alignment*, *Line Spacing*, *Insert*. With this fixed order, the overall MRR on the test set was only 0.3556.

D. Implementation Details

In this paper, we implemented NVMouse as a Microsoft Visual Studio Tools for Office (VSTO) Add-in[2]. To redefine mouse functions, we used the publicly available `MouseKeyHook` library[3] to capture mouse events and implemented custom event handlers to respond to these events. To access the document and application metadata, extract contextual features, and also to apply the selected application controls, we leveraged the services of the Interop Assembly. The Interop Assembly gave us access to application features and associated controls, as well as document content and its structure. Microsoft TTS was used to narrate the Feature Menu contents in response to user actions.

V. Evaluation

A. Participants

We recruited 14 participants (7 female, 7 male) who were completely blind, through local mailing lists and word-of-mouth (Table III). The participants varied in age between 31 and 63 ($M = 45.79$, $SD = 10.7$). All participants stated that they were either blind by birth or lost eyesight at a very young age (less than 10 years old). None of the participants had any motor impairments that affected their physical interaction with keyboard and mouse. Our inclusion criteria required that the participants be proficient with the JAWS screen reader and the Microsoft Word application on Windows platform.

B. Apparatus

The study was conducted using ASUS ROG GU501 laptop with Windows 10, Microsoft Word, JAWS screen reader, and NVMouse installed. An external standard keyboard and a wireless mouse were connected to the laptop. While NVMouse could have been interfaced with a touchpad instead of a mouse, we chose a mouse because in the pre-study interviews, touchpad was not preferred by all participants as they could not rest their hands comfortably on it before doing gestures, unlike keyboard and mouse.

[2]https://docs.microsoft.com/en-us/visualstudio/vsto/office-solutions-development-overview-vsto?view=vs-2017

[3]https://github.com/gmamaladze/globalmousekeyhook

TABLE III
Participant demographics for the user study.

ID	Gender	Age	Screen Reader	Word Usage Frequency
P1	F	31	JAWS, NVDA	Daily
P2	F	46	JAWS, NVDA	5 days a week
P3	M	60	JAWS	3 days a week
P4	M	39	JAWS, VoiceOver	Daily
P5	M	54	JAWS	2 days a week
P6	M	44	JAWS, VoiceOver	5 days a week
P7	F	56	JAWS	2 days a week
P8	F	45	JAWS, NVDA, System Access	5 days a week
P9	M	35	JAWS	2 days a week
P10	F	32	JAWS, System Access	Daily
P11	M	63	JAWS	2 days a week
P12	M	56	JAWS	1 day a week
P13	F	46	JAWS	3 days a week
P14	F	34	JAWS, VoiceOver	Daily

C. Design and Procedure

The main goal of the study was to evaluate how easily and quickly the participants could do different word-processing activities with NVMouse. Specifically, the participants did the following tasks: (i) **Task 1**: Find a pre-specified formatting command; (ii) **Task 2**: Read and delete all the comments in a document (all pre-specified); (iii) **Task 3**: Find and correct the typos in a pre-specified document; and (iv) **Task 4**: Navigate a predefined document and answer a question relevant to the document. Under a within-subjects design, the participants were asked to do the tasks under the following two conditions: (i) **Screen reader**: the participants used only the standard JAWS keyboard shortcuts; and (ii) **NVMouse**: the participants used only the mouse interface. In order to minimize learning effect, the ordering of tasks and conditions were counterbalanced using the Latin Square method [17].

For Task 1, the following two commands were chosen: (a) Set `Text Highlight Color` to 'Dark Blue'; (b) Select 'Heading 2' as `Style`. For Task 2, we created two 2-page documents each with 5 comments. For Task 3, we created two 2-page documents each with 5 typos at similar locations. For Task 4, we created two well-structured same-length documents from Wikipedia articles. Specifically, we chose two articles (each being 10 pages): (a) New York City; (b) Los Angeles.

Before starting the study, the participants were given enough practice time (~10 minutes) to familiarize themselves with the keyboard, and customize JAWS and Word settings to suit their preferences. Each study lasted for 2.5 hours, and all conversations during the study were in English.

Measurements. During the study, we logged all screen-reader keystrokes and mouse actions. Audio and computer-screen activities were also recorded using the *Open Broadcaster Software* for further analysis. We measured the task completion time and the number of shortcuts or mouse actions. At the end of the study, we administered the System Usability Scale (SUS) [18], NASA Task Load Index (NASA-TLX) [19], and an exit interview to collect subjective feedback.

TABLE IV
STATISTICS FOR TASK COMPLETION TIME AND NUMBER OF USER ACTIONS. THE BEST RESULTS ARE IN BOLD. THE WILCOXON SIGNED-RANK TEST RESULTS ARE ALSO SHOWN.

Task Completion Time (in seconds)

Task	Screen Reader				NVMouse				Significance Test Result
	μ	MD	Max	Min	μ	MD	Max	Min	
T1	214.5	220	472	45	**59.9**	60	95	39	Y(W=0<21)
T2	334.4	323.5	541	181	**122.7**	114	181	89	Y(W=0<21)
T3	478.7	452	865	110	**150.2**	133.5	310	74	Y(W=0<21)
T4	376	350	880	110	**99.5**	96.5	186	56	Y(W=0<21)

Number of Shortcuts/Mouse Actions

Task	Screen Reader				NVMouse				Significance Test Result
	μ	MD	Max	Min	μ	MD	Max	Min	
T1	47.1	47.5	66	28	**20.6**	21	25	16	Y(W=0<21)
T2	157.2	159	229	84	**35.2**	34	51	25	Y(W=0<21)
T3	146.5	146	186	100	**29.8**	30	36	23	Y(W=0<21)
T4	275.1	234.5	391	193	**55.7**	54	68	44	Y(W=0<21)

D. Results

1) Evaluation of the NVMouse Interface: Table IV presents the statistics for both the task completion time and the number of shortcuts/mouse actions for two study conditions, as well as the outcomes of statistical tests determining if the difference between the measures of two study conditions were statistically significant.

With NVMouse, all participants consistently performed better in all tasks, and the overall difference in both time taken and number of shortcuts/mouse actions between the two study conditions were statistically significant for all tasks. For Task 1, i.e., formatting, the participants struggled to properly navigate the complex ribbon structure with the screen reader, and find the target command. Ten participants, on at least one occasion, unintentionally pressed incorrect shortcuts that moved the focus away from ribbons, and therefore they had to repeat the command-search task by navigating the ribbons again from the beginning. No such accidental context switches were observed while using NVMouse. Also, since the task involved just finding a command, instead of applying it on some highlighted text, the prediction model did not contribute to the performance improvement; the default manually specified command order (mimicking the `Home` ribbon) was provided in the Formatting feature group. Therefore, all improvements are attributed only to the mouse interface.

For Task 2, with the screen reader, almost all (12) participants did not exactly know where to find the comments at the beginning, and therefore spent considerable time exploring the application. During the post-study interviews, they stated that although they occasionally collaborate with others, they find it difficult to remember the shortcut path to comments as well as changes. For Task 3, with the screen reader, all participants started off by manually checking each word, before searching for the application control that automatically moves the focus between typos. This contributed significantly to the completion time. Even with NVMouse, five participants started manually inspecting word-by-word, before switching to the mouse interface. For Task 4, with the screen reader, the participants mostly navigated line by line, while sometimes making fast hotkey-presses to quickly navigate through irrelevant document sections. As shown in Table IV, this turned out to be much slower than hierarchical content navigation enabled by NVMouse.

2) Subjective Evaluation: **System Usability Scale (SUS).** For the standard SUS questionnaire [18], the participants rated positive and negative statements about each study condition on a Likert scale from 1 for strongly disagree to 5 for strongly agree, with 3 being neutral. Overall, we found a significant difference in the SUS scores between screen reader ($\mu = 58.57$, $\sigma = 16.73$) and NVMouse ($\mu = 85.35$, $\sigma = 4.41$) conditions (Wilcoxon signed-rank test, $W = 1 < 21$, $p < 0.001$, $n = 14$).

NASA-TLX. NASA-TLX [19] is widely used for assessing perceived task workload (expressed as a value between 0 and 100, with lower values indicating better results). Overall, we found a significant difference in the NASA-TLX scores between screen reader ($\mu = 61.49$, $\sigma = 13.77$) and NVMouse ($\mu = 18.80$, $\sigma = 2.55$) conditions (Wilcoxon signed-rank test, $W = 0 < 21$, $p < 0.001$, $n = 14$).

3) Qualitative Feedback: All participants stated that NVMouse's input actions were much simpler, intuitive, and easier to remember and perform, compared to the screen-reader keyboard shortcuts. Seven participants (P2, P3, P5, P6, P9, P11, and P12) stated that they frequently get confused between the screen reader's web shortcuts and the word-processing shortcuts, and therefore make mistakes. However, they indicated that they would never run into such an issue with NVMouse. Six participants (P2, P6, P7, P9, P10, and P12) expressed that NVMouse allowed them to do actions with just one hand, whereas the keyboard interface often required them to use two hands to press complex key combinations (e.g., *INSERT+F7*) as shortcuts, which occasionally caused them to make unintentional mistakes when the keys were far apart from each other on the keyboard; such problems will not occur with the mouse interface of NVMouse.

Twelve participants (except P1 and P4) noted that in contrast to the screen reader, the NVMouse interface has clear "entry" and "exit" points when accessing controls. They stated that with keyboard there are multiple ways/shortcuts which one can use to enter, navigate, or exit the ribbons, thereby increasing the likelihood of unintentionally skipping certain controls that can only be accessed through specific shortcuts. They mentioned that such an issue will not arise with NVMouse, as there was only one way to access, navigate, and exit the feature menu.

VI. DISCUSSION

Limitations. In this paper, the potential of the "repurposed" mouse was validated only for word-processing applications. However, the underlying concepts and methodologies easily generalize to other applications as well, since most applications follow a typical GUI design pattern like word processors, i.e., having a main work-area surrounded by many application

features and commands, which is more suitable for visual "point-and-click" interaction for sighted users.

Also, the current NVMouse prototype relies on Microsoft's Interop Assembly to access the application metadata, and therefore it is not directly adaptable for non-Microsoft applications. However, the modular structure of NVMouse easily lets it replace Interop with other alternatives such as UI Automation accessibility framework [20], thereby enabling NVMouse to work with other applications. Lastly, the prediction model for dynamically reordering commands needs to be separately defined for each application. However, this is expected since different applications have different purposes, and therefore the notion of 'local user context' varies across applications.

Future work – beyond word processing. As mentioned earlier, NVMouse can be easily adapted for other applications besides word processors including productivity tools such as Excel, PowerPoint, Google Docs, Google Sheets, etc., by simply mapping the corresponding application features and commands to the NVMouse's Feature Menu. As for a prediction model, the idea of exploiting 'local context' to make command prediction, generalizes to other applications as well. For instance, in Excel, a group of cells surrounding the focused cell, can be considered as context, and attributes such as content type, formatting style, background color, presence of formulas, etc., can be leveraged to train a prediction model. For example, if the content of the surrounding cells is a mixture of formulas and bold or italicized words (such as total, average, etc.), and the cell to the immediate left or top of the focused cell has bold or italicized words, then it is very likely that the user may want to insert a formula. Similarly, in PowerPoint, in each slide, objects such as shapes, pictures, text boxes, etc., surrounding the current object in focus, can be defined as context, from which features (e.g., object properties and type details) can be extracted and used to train a prediction model.

VII. Conclusion

This paper introduces a non-visual scroll-and-click version of a visual point-and-click mouse input device, to break blind users' sole reliance on a keyboard for accessing various application features using screen readers. The paper also provides experimental evidence of the promise of NVMouse in enabling blind computer users become much more productive with word processors. It is anticipated that further research on this new interaction paradigm centered on a repurposed mouse will hopefully usher similar productivity gains for blind users in general with any computing application.

Acknowledgments

This work was supported by NSF Awards: 1805076, 1936027, NIH Awards: R01EY026621, R01EY030085, R01HD097188, NIDILRR Award: 90IF0117-01-00.

References

[1] Freedom Scientific, "Jaws ® – freedom scientific," http://www.freedomscientific.com/products/software/jaws/, 2020.

[2] Apple Inc., "Vision accessibility - mac - apple," https://www.apple.com/accessibility/mac/vision/, 2020.

[3] NV Access, "Nv access," https://www.nvaccess.org/, 2020.

[4] E. C. Bell and N. M. Mino, "Employment outcomes for blind and visually impaired adults," *Journal of Blindness Innovation and Research*, vol. 5, 01 2015.

[5] K. Wang, L. G. Barron, and M. R. Hebl, "Making those who cannot see look best: Effects of visual resume formatting on ratings of job applicants with blindness." *Rehabilitation psychology*, vol. 55, no. 1, p. 68, 2010.

[6] I. Abu Doush, E. Pontelli, D. Simon, T. C. Son, and O. Ma, "Making microsoft excel™: Multimodal presentation of charts," in *Proceedings of the 11th International ACM SIGACCESS Conference on Computers and Accessibility*, ser. Assets '09. New York, NY, USA: ACM, 2009, pp. 147–154. [Online]. Available: http://doi.acm.org/10.1145/1639642.1639669

[7] C. Asakawa and T. Itoh, "User interface of a home page reader," in *Proceedings of the Third International ACM Conference on Assistive Technologies*, ser. Assets '98. New York, NY, USA: ACM, 1998, pp. 149–156. [Online]. Available: http://doi.acm.org/10.1145/274497.274526

[8] E. Więcek-Janka, M. Papierz, M. Kornecka, and M. Nitka, "Apple products: A discussion of the product life cycle," in *4th International Conference on Management Science and Management Innovation*, vol. 31, 2017, pp. 159–164.

[9] S. M. Billah, V. Ashok, D. E. Porter, and I. Ramakrishnan, "Speed-dial: A surrogate mouse for non-visual web browsing," in *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*, ser. ASSETS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 110–119. [Online]. Available: https://doi.org/10.1145/3132525.3132531

[10] R. Khurana, D. McIsaac, E. Lockerman, and J. Mankoff, "Nonvisual interaction techniques at the keyboard surface," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3173574.3173585

[11] G. Mori, M. C. Buzzi, M. Buzzi, B. Leporini, and V. M. R. Penichet, "Making "google docs" user interface more accessible for blind people," in *Advances in New Technologies, Interactive Interfaces, and Communicability*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 20–29.

[12] M. Connolly, C. Lutteroth, and B. Plimmer, "Document resizing for visually impaired students," in *Proceedings of the 22nd Conference of the Computer-Human Interaction Special Interest Group of Australia on Computer-Human Interaction*, 2010, pp. 128–135.

[13] M. C. Buzzi, M. Buzzi, B. Leporini, and G. Mori, "Designing e-learning collaborative tools for blind people," *E-Learning-Long-Distance and Lifelong Perspectives (2012)*, pp. 125–144, 2012.

[14] L. Morales, S. M. Arteaga, and S. Kurniawan, "Design guidelines of a tool to help blind authors independently format their word documents," in *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '13. New York, NY, USA: ACM, 2013, pp. 31–36. [Online]. Available: http://doi.acm.org/10.1145/2468356.2468363

[15] J. Schoeberlein and Y. Wang, "Usability evaluation of an accessible collaborative writing prototype for blind users," *Journal of Usability Studies*, vol. 10, no. 1, pp. 26–45, 2014.

[16] M. M. Waqar, M. Aslam, and M. Farhan, "An intelligent and interactive interface to support symmetrical collaborative educational writing among visually impaired and sighted users," *Symmetry*, vol. 11, no. 2, p. 238, 2019.

[17] J. V. Bradley, "Complete counterbalancing of immediate sequential effects in a latin square design," *Journal of the American Statistical Association*, vol. 53, no. 282, pp. 525–528, 1958.

[18] J. Brooke *et al.*, "Sus-a quick and dirty usability scale," *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.

[19] S. G. Hart and L. E. Staveland, "Development of nasa-tlx (task load index): Results of empirical and theoretical research," in *Human Mental Workload*, ser. Advances in Psychology, P. A. Hancock and N. Meshkati, Eds. North-Holland, 1988, vol. 52, pp. 139 – 183. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166411508623869

[20] R. Haverty, "New accessibility model for microsoft windows and cross platform development," *SIGACCESS Access. Comput.*, no. 82, p. 11–17, Jun. 2005. [Online]. Available: https://doi.org/10.1145/1077238.1077240