Boston University Computer Science Technical Report No. BUCS-TR-2011-009

Menu Controller: Making Existing Software More Accessible for People with Motor Impairments

Isaac Paquette, Christopher Kwan, and Margrit Betke
Image and Video Computing Group
Department of Computer Science
Boston University
111 Cummington Street
Boston, MA 02215
{paquette, ckwan, betke}@cs.bu.edu

ABSTRACT

Menu Controller was developed to make existing software more accessible for people with severe motor impairments, especially individuals who use mouse-replacement input systems. Windows applications have menus that are difficult to access by users with limited muscle control, due to the size and placement of the menu entries. The goal of Menu Controller is to take these entries and generate customizable user interfaces that can be catered to the individual user. Menu Controller accomplishes this by harvesting existing menu items without needing to change any existing code in these applications and then by displaying them to the user in an external toolbar that is more easily accessible to people with impairments. The initial challenge in developing Menu Controller was to find a method for harvesting and re-displaying menu items by using the Windows API. The rest of the work involved exploring an appropriate way for displaying the harvested menu entries. We ultimately chose an approach based on a two-level sliding toolbar. Experiments with a user with severe motor impairments, who used the Camera Mouse as a mouse-replacement input system, showed that this approach was indeed promising. The experiments also exposed areas that need further research and development. We suggest that Menu Controller provides a valuable contribution towards making everyday software more accessible to people with disabilities.

Categories and Subject Descriptors

K.4.2 [**Computers and Society**]: Social Issues—*assistive technologies for persons with disabilities*; H.1.2 [**Models and Principles**]: User/Machine Systems—*human factors*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*interaction styles (e.g., commands, menus, forms, direct manipulation)*

General Terms

Human Factors

Keywords

Accessibility, Assistive software, Camera Mouse, Human computer interaction, User interfaces, Video-based interfaces

1. INTRODUCTION

1.1 Motivation

A key concept in enabling computer control for people with disabilities is "transparent access which allows them to use any commercial software application through whatever input device they employ" [24]. However, there still exists a huge gap between the control afforded by specialized input devices and the accessibility provided by commercial applications. Although creating custom application software for users with disabilities, when software with the same functionality already exists for able-bodied users, is not always the most feasible or economical solution [24], sometimes it is the only usable solution for these users.

The inspiration for Menu Controller came from existing research on methods of providing access to specific Windows applications to people with severe motor impairments. These efforts include, but are not limited to, IWebExplorer [23], which provides access to the web by embedding a web browser control and providing an on-screen keyboard to the user; HAILbrowser, which provides toolbars with large buttons to access links on web pages within an embedded browser that serves up mobile versions of web pages; and HAILtwitter, which uses HAIL's toolbar system to control a user's Twitter account. In the paper describing the HAIL paradigm, the authors indicated that future projects were in the planning stage, which would use HAIL to provide access to "an email client, news-feed aggregator, Facebook interface, and media player application" [19]. While these efforts would provide a richer and more-easily-customizable experience for these specific applications, much developer effort is required to provide this functionality. It became apparent that having a generic approach to access a subset of the functionality of existing Windows applications, while other more application-specific projects are being worked on, would give users the ability to start using these applications now while they wait

for the richer implementations of their favorite programs to become reality.

1.2 Camera Mouse Interface

Menu Controller was developed for individuals with severely limited muscle control and tested with a camera-based mouse-replacement system, the Camera Mouse. The Camera Mouse [5] is software that was developed at Boston College and Boston University and is freely available on the web for download [7]. It is designed for individuals with severe motor impairments who cannot use their hands to operate a computer mouse. The software works by tracking a feature on the user's face with a web camera, allowing him or her to control the mouse pointer with head movements. A user clicks by "dwelling" over the item to be selected - placing the mouse pointer within a small radius of the item for a set amount of time, e.g. one second.

Using Camera Mouse, individuals with disabilities can operate specially designed software such as Eagle Aliens and Eagle Paint [4]. It is also possible for these users to interact with existing software that is not specifically designed for people with severe motor impairments. However, such software typically has menu-based user interfaces involving drop-down menus with small entries that are close to one another (Figure 2). These interfaces are usually difficult to use with Camera Mouse or other mouse-replacement systems [12, 20] because of the degree of precision they require in order to choose individual entries. For users of mouse-replacement systems with and without disabilities alike, it is difficult to "dwell" within such small areas, so it is common for users to miss target entries or to accidentally hit neighboring entries.

In designing specialized input devices for users with disabilities, it is important for users to be able to use the devices independently, with minimal assistance from caregivers [24]. The same is true for software. If it was easier for users with disabilities to access menu entries, which often contain key functions or customizations, they could rely less on assistance from caregivers and have a higher degree of independence when using applications.

1.3 Related Work

Our work relates to general work in input and output redirection and reverse engineering of user interfaces. Two projects utilizing redirection on the Windows platform are *mudibo* [16], which can simultaneously duplicate dialog boxes across multiple monitors, allowing a user to interact with the dialog in any location and *WinCuts* [27], which allows a user to replicate portions of existing windows and interact with them as new independent windows. Stuerzlinger *et al.* [26] developed *User Interface Façades,* a system for adapting existing user interfaces in general. Their system uses direct manipulation techniques and requires no programmatic changes to the existing applications. It provides users abilities including: the ability to create new user interfaces using duplicated screen regions, the ability to add holes to user interfaces in order to overlay applications on top of one another, and most relevantly, the abilities to modify the interaction behavior of existing user interface widgets or to replace them entirely with new ones.

There are also projects that achieve redirection and reverse engineering of user interfaces with image processing rather than API's or user interface toolkits. The *SegMan* system [25] translates pixel-level input, such as the appearance of user interface components, into objects and symbols for cognitive models, so that the models can interact with existing Windows applications. Hurst *et al.* [14] improve upon the Microsoft Active Accessibility API's [1] ability to detect the location and size of user interface targets by developing a hybrid approach that combines the API with machine learning and computer vision techniques. Finally, *Prefab* [10] is a system that uses a pixel-based approach, independent of specific user interface toolkits or platforms, to reverse engineer the user interface structures of existing applications. Using input and output redirection, *Prefab* can then modify the apparent behavior of these interfaces or even implement new advanced behaviors.

Our work also relates to work that addresses the targeting difficulties some users experience. Similar to how we address the difficulties that Camera Mouse users have in selecting small, closely grouped menu entries, Worden *et al.* [30] address the difficulties that older adults have in making small mouse movements and clicking on small targets. Instead of trying to modify the interface layouts of existing applications, the authors developed two new interaction techniques that operate within existing layouts: *area cursors* – cursors with larger than normal activation areas and *sticky icons* – icons that automatically reduce the cursor's gain ratio when it is on them, making it easier for the mouse pointer to stop or "stick" on the icon. The *Bubble Cursor* [13] is an improvement on the *area cursor* such that it dynamically resizes its activation area so that only one target is selectable at any time. Hurst *et al.* [15] also address the problem of making user interface targets easier to select. They use an adaptive pointing technique where small forces are associated with past clicks. Frequently clicked-on areas accumulate a *pseudo-haptic* magnetic field that draws the mouse pointer to them in a way similar to *sticky icons.*

Our work is also related to projects in creating tools that can provide access to or augment existing applications. Akram *et al.* [2] developed an application mediator to give Camera Mouse users a greater degree of autonomy when launching applications or switching between tasks. Their system has an accessible parent menu that provides access to a fixed set of application tools, including a text-entry program, web browser and music player. Another accessibility project that provides a generic approach for accessing more than one application is *Johar* [3]. It provides a mechanism that developers of applications can implement that will allow external user interfaces to manipulate their applications. However, Johar can only be used for applications that are explicitly designed to cater to the Johar interface. Olsen *et al.* described an architecture for creating interface *attachments* – small independent programs, such as a text searcher or a spell checker, that can augment the functionality of a variety of applications. Their implementation involves intercepting components of a Java user interface toolkit in order to access the visual information that the applications display on screen [22].

Finally, our work also relates to similar work in designing

specialized user interfaces for users with severe motor impairments. SUPPLE is a system that automatically generates personalized user interfaces for individual users based on their motor capabilities [11]. The HAIL (Hierarchical Adaptive Interface Layout) model presents specifications for the design of user interfaces that can change and adapt to users with severe motion impairments [19]. The approaches of both SUPPLE and HAIL look at generating user interfaces at a programmatic level; creating more usable and adaptive interfaces by creating new applications. In Menu Controller, we address the different but related problem of generating user interfaces for software that already exists, for which we do not have access to modifying the underlying source code. We look at how we can transform these already implemented interfaces to make them more usable and customizable to the needs of users with severe motor impairments.

2. METHODS

2.1 Challenges

Early on in the process of developing Menu Controller, research was done to see if it was possible to control the menu items of a Windows application from an external program. After trying many different options, we discovered that the Windows API [28] is designed to allow developers to simulate any action that a user can accomplish with a mouse or keyboard. Windows messages are sent to individual items on a window (such as a menu item), and these items respond to these messages in the same way that they would respond to an actual action performed directly by the user on that item, e.g. a click. This gives a programmer the power to control almost any aspect of any window without knowledge of the inner workings of the window itself.

The next step was to experiment with the MenuAPI [21]. This API is a subset of the WindowsAPI that deals specifically with Windows Menus. Once we figured out which methods we would need from the API, the work from that point forward involved testing Menu Controller on several Windows applications. Of particular interest was our testing with Windows Media Player [29], as we wanted to try to get one of the programs mentioned in the HAIL paper [19] working with Menu Controller (Figure 5).

Once it was determined that the WindowsAPI would give us control of the menu entries, the next step was to decide how they should be re-rendered by Menu Controller. We ultimately decided on an approach based on a *sliding toolbar*, a design that was developed for the Camera Canvas project [17, 18].

2.2 User Interface

The way in which Menu Controller re-renders the menu entries of an application is based on the user interface of a new version of Camera Canvas [18] (Figure 1), an image editing program for people with severe motor impairments, designed for use with the Camera Mouse. Camera Canvas uses a toolbar with large buttons to provide users with motor impairments easy access to the functionality of the program. We follow a similar approach in Menu Controller: we re-render application menu entries in large button form in a toolbar at the top of the screen.

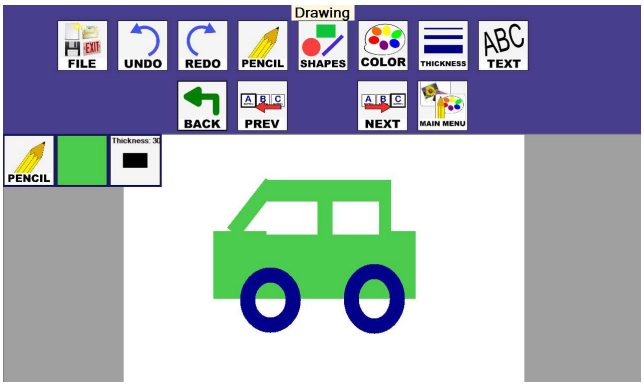

Figure 1: The Camera Canvas [18] image editing program with its *sliding toolbar* user interface at the top of the screen. All buttons in the top row (between "FILE" and "TEXT") move to the left by one button position if the "PREV" button is selected. Similarly, all buttons move to the right by one position if the "NEXT" button is selected.

When a user navigates to an application with a menu, the user first sees the root entries of the menu. Similarly, when Menu Controller first encounters an application, it displays the root menu entries as a sequence of large buttons (Figure 3). When a user clicks on an entry in the root menu of the application, a submenu is typically displayed (Figure 2). The same behavior is achieved in Menu Controller. When one of root buttons is selected, a list of buttons for the associated submenu replaces the root buttons, and so on. When a user is navigating a menu, its submenus disappear when the user clicks off of the menu. At this point the user again sees only the root menu entries. Menu Controller behaves in a similar way: when a user clicks off of Menu Controller and onto the main window of the application, Menu Controller again renders the root menu entries of that application.

The sequence of large buttons displayed by Menu Controller have a *sliding* functionality. The toolbar has two arrow buttons: a "Prev" and a "Next" button that enable the user to "slide" the toolbar across the screen, that is, collectively moving the positions of the menu buttons on the screen. The aim of the sliding functionality is to help users who cannot reach certain areas of the screen. For example, if a user cannot reach a button at the far left of the screen, the user can click on the arrow buttons and continually slide the toolbar, moving the button towards the middle of the screen, until the button is within the user's reach. The direction of sliding is from the perspective of looking at the space between the two arrow buttons in the center of the screen. The "Prev" arrow button causes the button in the previous position to the center to slide to the center, and the "Next" arrow causes the button in the next position to the center to slide to the center.

2.3 How Menu Controller Works

When Menu Controller is run, a timer is started and a "tick" event from the timer is handled by Menu Controller once every second. The handler first makes a call to the `GetForegroundWindow` WindowsAPI method, which returns a handle to the window that currently has focus. If no window

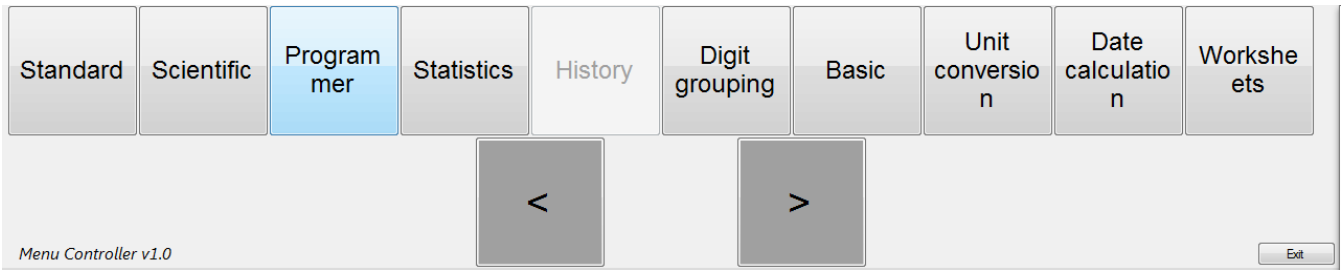

Figure 3: Menu Controller's re-rendering of the View submenu of the Microsoft Windows 7 Calculator [6] program.

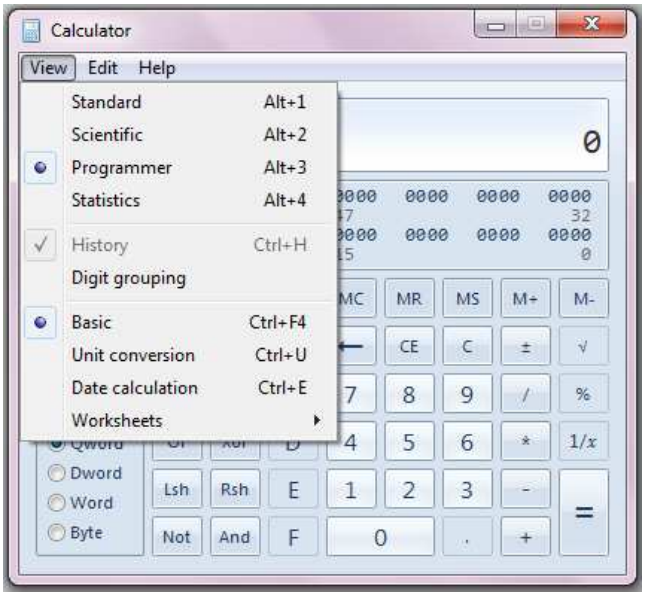

Figure 2: The View submenu of the Microsoft Windows 7 Calculator [6] program. The menu entries are small and closely grouped together, making them difficult to access using the Camera Mouse.

currently has focus, i.e., all windows are minimized, Menu Controller does nothing and simply waits for the next timer tick. If a window is found, Menu Controller then makes a call to the GetMenu API function, passing it the handle to the in-focus window, which returns another handle that points specifically to the menu of that window. (Note: If the returned handle is 0, this means that the window either does not have a menu, or that the menu is not the type of menu that can be accessed using the MenuAPI calls [21].) Once a valid handle to a menu is obtained, the first level of the menu is read (i.e. the part of the menu visible to the user prior to clicking on any menu items), and information about each first level menu entry is stored in a list. This list is then used to dynamically create the buttons that are displayed to the user within Menu Controller, the text of which is retrieved from the menu items themselves using `GetMenuString` [21]. (Note that, while creating each button, within each button a handle to the menu itself is stored, which was obtained along with the index of the menu item the button is associated with.) At this point, the user sees the first-level menu items in the Menu Controller toolbar.

When the user clicks on buttons in the toolbar, the same event handler is initiated for all of the buttons. What differentiates the buttons from one another from the perspective of the button click handler is the data that Menu Controller previously stored with each button, namely the menu handle and the index of the menu associated with the given button. With these two pieces of information, Menu Controller makes a further WindowsAPI call to `GetSubMenu` [21] to determine if the item is a submenu or an actual item that needs to be clicked. If the former, Menu Controller follows the same steps outlined above to read the menu items of the submenu, and dynamically create buttons to be displayed, but this time for the submenu. If the latter, the appropriate information, in this case the handle to the menu along with the index of the item to be clicked, is sent to the appropriate WindowsAPI methods to simulate the clicking of the item. These methods include `GetMenuItemID` [21] and `SendMessage`. `GetMenuItemID` provides one of the parameters for SendMessage, a generic WindowsAPI method that is used to send all types of messages to any window. (We note here that the call to SendMessage has the exact effect from perspective of the Windows operating system as if the user had clicked on the menu item directly.)

2.4 Two Types of Menus in Windows

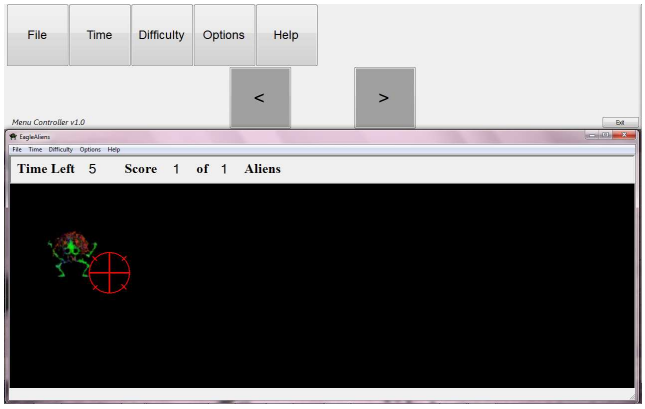

Figure 4: Menu Controller displaying the top level menu of the Eagle Aliens [4] game designed for Camera Mouse [5].

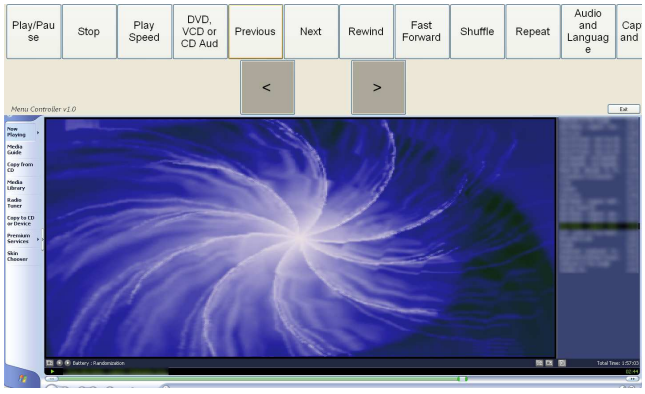

Figure 5: Menu Controller displaying the Play submenu in Windows Media Player 9 [29].

There are two standard types of menus in Windows. One is the type that is accessible using the MenuAPI calls [21], and the other is based on a more recent technology that Microsoft developed called Active Accessibility [1]. Most legacy applications use the former type, while newer applications take advantage of the Active Accessibility technology, which was developed by Microsoft to allow software developers to write accessibility tools that can interact with their applications.

Menu Controller currently only supports the MenuAPI-accessible menus, which means that it can only support a subset of Windows applications. In our testing, however, we found that there are many useful programs that use the older menu style, and concluded that Menu Controller can handle a large subset of Windows applications.

3. EXPERIMENTS INVOLVING A USER WITH MOTOR IMPAIRMENTS

We recently conducted experiments with a non-verbal individual with severe cerebral palsy and quadriplegia (Figure 6). The participant has used the Camera Mouse input system in the past. For our experiments, we asked the participant to use Eagle Aliens, a game where the user moves the mouse pointer around the screen to "shoot aliens" (Figure 4), and Eagle Paint, a freeform line drawing program. Both are popular programs designed for use with Camera Mouse [4] that the participant was already familiar with. We explained that the goal of Menu Controller is to allow a user to operate more of the program's features by him or herself. The participant especially liked playing Eagle Aliens and seemed excited at the prospect of being able to start a new game or adjust the difficulty level by himself.

Functionality, such as starting a new game or adjusting the difficulty level, is only available via the menu of Eagle Aliens and could not be accessed by the participant. Only when Eagle Aliens was used together with Menu Controller could the participant access the functionality.

When playing Eagle Aliens, the participant was able to use Menu Controller to open the File menu and start a new game and adjust the time limit and difficulty settings of the game.

We observed that it was difficult for the participant to reach buttons that Menu Controller displayed in the top left corner of the screen. We explained how the arrow buttons allowed him to move the buttons toward the center of the screen to be more within his reach. After a couple of explanations on how the movement worked, selecting the correct arrow became more intuitive for the participant. The participant seemed to like the idea of the arrow buttons, but due to their placement, they seemed to do more harm than good. Because of their close proximity to the menu buttons, the user often had to pass over an arrow button to get to a menu button. Doing this would sometimes cause the menu buttons to slide, shifting his intended target. We learned that moving the arrow buttons farther from the menu buttons, or even to a different area of the screen, so that they are not as easily triggered by mistake, is a much needed change.

Although the participant was able to hit some of the buttons, in general it was difficult for the user to make movements to reach the top of the screen. To try to help the user, we re-initialized Camera Mouse and adjusted the mouse movement gain settings, but the user still had difficulty. It would be beneficial for this user if the Menu Controller could be moved to a different area of the screen. Also, the buttons on the Menu Controller were too close together so when the user tried to click on a button he would often click neighboring buttons by mistake. It would be very helpful to be able to adjust the button size and space between buttons at runtime.

When the participant clicked on the appropriate Menu Controller button to adjust the difficulty level or time parameter of the game, no feedback was provided to show that the click was successful, and so the participant would continue trying to click the same button over and over. Menu options to adjust settings, such as "Difficulty" or "Time," present the user with a list of options, only one of which can be selected. In the original Windows style menu of Eagle Aliens (outside of Menu Controller), when the user clicks to select an option, a checkmark appears next to that option to signify that the option is selected (Figure 2). However, in Menu Controller there is not yet an equivalent means to provide

Figure 6: A user with motor impairments using Menu Controller with the Eagle Aliens game.

this type of feedback. A possible future enhancement would be to change the button appearance in some way such as by drawing a dark border around the selected item, so as to provide the user with this feedback.

Several times during the experiment, the user accidentally clicked outside the application window, causing Menu Controller to automatically hide itself. It would be beneficial if, when started, Menu Controller automatically resized itself and the application window to take up the whole screen to prevent this from happening.

In using Eagle Paint, the participant was able to launch some menu items such as changing the background color to black, but still had the same problems as when using Eagle Aliens. At this point in the experiments the user was also feeling fatigued so it was even more tiring for him to make the movements required to reach the Menu Controller at the top of the screen.

Eagle Aliens and Eagle Paint both require the spacebar key to be pressed in order to start the main part of the program (causing the aliens to appear or the lines to start being drawn). Because of that, the user still needs the assistance of a caregiver in order to use the application. It would be beneficial if we added buttons of common keyboard commands such as "Space", "Enter" or "Tab" to the toolbar so that the user can activate programs requiring these keys by himself.

We learned a lot from the participant of our experiments and his interaction with Menu Controller and the two application programs. It was very encouraging to see his positive reaction to the software despite the difficulties he had in using it. We now know the areas where we need to make improvements and are even more motivated to making the program as usable as possible.

4. DISCUSSION AND CONCLUSIONS

To address the difficulties that users with motor impairments have in accessing the menu entries of existing applications, we developed Menu Controller, a tool that can harvest the menu entries of existing applications and present them to a user in a form that is more accessible and usable. We conducted experiments with a user with severe motor impairments and observed that Menu Controller allowed him to access previously unavailable functionality in applications that he already uses. The studies conducted with this user helped us identify areas of improvement and directions for future research.

Our future work involves incorporating what we learned from our most recent user study in order to make Menu Controller more usable. We hope to make the appropriate changes and conduct another session with the user from that study. We will also be conducting more user studies with additional users with and without disabilities and improving the software based on their feedback.

Eventually, we hope to make the way Menu Controller re-renders menus highly customizable so that each user can tailor it specifically to his or her needs and abilities. Currently, Menu Controller re-renders menus in only a horizontal orientation at the top of the screen with large buttons, as shown in Figure 3 for the example of the Microsoft Windows 7 Calculator. In the future, users should be able to place the toolbar in different areas of the screen, change its orientation, and adjust the size and spacing of the buttons. The goal is for a user to be able to use Menu Controller with several applications and have a consistent, customized, and accessible user experience throughout. The great variety in software user interfaces, often presenting difficulties even for users without disabilities, bring even more frustration for users with disabilities who may have to customize their input devices for every application [24]. With Menu Controller, our hope is to alleviate some of this frustration by reducing the number of different application user interfaces that users will have to customize their input devices against.

We are also investigating how to handle the newer Microsoft Active Accessibility user interface technology [1], which will make Menu Controller usable with the vast majority of Windows applications that have a menu.

Extensive capture and manipulation techniques for user interfaces were developed in the mentioned works. In our work, we use more basic techniques, but focus specifically on users with motor impairments. Utilizing similar approaches to the ones mentioned, we could improve Menu Controller to handle other types of user interface widgets in addition to menus. Currently, our focus is only on the Windows environment since Camera Mouse is presently only available to Windows users. However, employing image processing techniques similar to the systems mentioned above [25, 14, 10] could allow us to develop an accessibility tool that is platform independent.

While the described pointer-targeting techniques may be helpful to users with motor impairments, they do not provide much added benefit when used with targets that are small and close together, such as those in menu entries. Using these techniques in conjunction with the user interface created by Menu Controller might increase their utility for these users.

Eventually, Menu Controller will be made freely available to the public for download [7]. We hope to allow the extensive

worldwide Camera Mouse user base [8] access to more applications and to encourage new users in trying Camera Mouse after seeing that Menu Controller may give them additional computing ability.

Studies have found that adoption of software to assist computer users with severe motor impairments is sometimes difficult. Dawe [9] found that the rate that some of this software is abandoned due to various factors (including cost and complexity) is estimated to be upwards of 35%. Our goal with Menu Controller is, therefore, to make it intuitive and easy to use in order to attract a user community that will find it useful while also simple to use. We also hope that Menu Controller's support for a number of existing Windows applications will make its adoption even more attractive.

5. ACKNOWLEDGEMENTS

We would like to thank the participant of our user study for his assistance and patience in helping us evaluate our software. We would also like to thank the members of the Image and Video Computing Group for their advice and feedback throughout the project. The authors gratefully acknowledge NSF funding (HCC grants IIS-0713229 and IIS-0910908).

6. REFERENCES

[1] Active Accessibility. Retrieved December 5, 2010, from `http://msdn.microsoft.com/en-us/library/aa291313%28VS.71%29.aspx`.

[2] W. Akram, L. Tiberii, and M. Betke. A customizable camera-based human computer interaction system allowing people with disabilities autonomous hands-free navigation of multiple computing tasks. In *Proceedings of the 9th conference on User interfaces for all*, ERCIM'06, pages 28–42, Berlin, Heidelberg, 2007. Springer-Verlag.

[3] J. H. Andrews and F. Hussain. Johar: a framework for developing accessible applications. In *Proceedings of the 11th international ACM SIGACCESS conference on Computers and accessibility*, Assets '09, pages 243–244, New York, NY, USA, 2009. ACM.

[4] M. Betke. Intelligent interfaces to empower people with disabilities. In H. Nakashima, J. C. Augusto, and H. Aghajan, editors, *Handbook of Ambient Intelligence and Smart Environments*. Springer Verlag, June 2009.

[5] M. Betke, J. Gips, and P. Fleming. The Camera Mouse: Visual tracking of body features to provide computer access for people with severe disabilities. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 10(1):1–10, Mar. 2002.

[6] Calculator - Windows 7 features - Microsoft Windows. Retrieved December 19, 2010, from `http://windows.microsoft.com/en-US/windows7/products/features/calculator`.

[7] The Camera Mouse website. `http://www.cameramouse.org`, 2010.

[8] Camera Mouse technology reaches 100,000th download milestone. `http://www.bc.edu/publications/chronicle/TopstoriesNewFeatures/features/cameramouse030410.html`, Mar. 2010.

[9] M. Dawe. Desperately seeking simplicity: how young adults with cognitive disabilities and their families adopt assistive technologies. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, CHI '06, pages 1143–1152, New York, NY, USA, 2006. ACM.

[10] M. Dixon and J. Fogarty. Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 1525–1534, New York, NY, USA, 2010. ACM.

[11] K. Z. Gajos, D. S. Weld, and J. O. Wobbrock. Automatically generating personalized user interfaces with Supple. *Artificial Intelligence*, 174:910–950, 2010.

[12] D. Gorodnichy, E. Dubrofsky, and M. Ali. Working with computer hands-free using Nouse perceptual vision interface. In *Proceedings of the International CRV Workshop on Video Processing and Recognition (VideoRec'07), Montreal, Candada*, Canada, May 2007. NRC.

[13] T. Grossman and R. Balakrishnan. The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '05, pages 281–290, New York, NY, USA, 2005. ACM.

[14] A. Hurst, S. E. Hudson, and J. Mankoff. Automatically identifying targets users interact with during real world tasks. In *Proceeding of the 14th international conference on Intelligent user interfaces*, IUI '10, pages 11–20, New York, NY, USA, 2010. ACM.

[15] A. Hurst, J. Mankoff, A. K. Dey, and S. E. Hudson. Dirty desktops: using a patina of magnetic mouse dust to make common interactor targets easier to select. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, UIST '07, pages 183–186, New York, NY, USA, 2007. ACM.

[16] D. R. Hutchings and J. Stasko. mudibo: multiple dialog boxes for multiple monitors. In *CHI '05 extended abstracts on Human factors in computing systems*, CHI '05, pages 1471–1474, New York, NY, USA, 2005. ACM.

[17] W.-B. Kim, C. Kwan, I. Fedyuk, and M. Betke. Camera canvas: Image editor for people with severe disabilities. Technical Report 2008-010, Computer Science Department, Boston University, May 2008.

[18] C. Kwan and M. Betke. Camera Canvas: Image editing software for people with disabilities. In *International Conference on Human-Computer Interaction, Orlando, Florida, July 2011.* To be published.

[19] J. Magee and M. Betke. HAIL: hierarchical adaptive interface layout. In K. M. et al., editor, *12^{th} International Conference on Computers Helping People with Special Needs (ICCHP 2010), Vienna University of Technology, Austria, Part 1, LNCS 6179*, pages 139–146. Springer-Verlag Berlin Heidelberg, July 2010.

[20] C. Manresa-Yee, J. Varona, F. J. Perales, F. Negre, and J. J. Muntaner. Experiences using a hands-free interface. In *Proceedings of the 10th International ACM SIGACCESS Conference on Computers and*

Accessibility, pages 261–262, New York, NY, 2008. ACM.

[21] Menus. Retrieved December 5, 2010, from `http://msdn.microsoft.com/en-us/library/ms646977%28v=VS.85%29.aspx`.

[22] D. R. Olsen, Jr., S. E. Hudson, T. Verratti, J. M. Heiner, and M. Phelps. Implementing interface attachments based on surface representations. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, CHI '99, pages 191–198, New York, NY, USA, 1999. ACM.

[23] M. Paquette, M. Betke, and J. Magee. IWeb Explorer: A web browser designed for use with an eye controlled mouse device. Master's thesis, Computer Science Department, Boston University, April 2005. 11 pp.

[24] F. Shein. Human interface design and the handicapped user. In *Proceedings of the Computer-Human Interaction Conference*, pages 292–293. ACM, 1986.

[25] R. St. Amant, M. O. Riedl, F. E. Ritter, and A. Reifers. Image processing in cognitive models with SegMan. In *Proceedings of the 11th International Conference on Human-Computer Interaction*, HCII 2005, 2005.

[26] W. Stuerzlinger, O. Chapuis, D. Phillips, and N. Roussel. User Interface Façades: towards fully adaptable user interfaces. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, UIST '06, pages 309–318, New York, NY, USA, 2006. ACM.

[27] D. S. Tan, B. Meyers, and M. Czerwinski. WinCuts: manipulating arbitrary window regions for more effective use of screen space. In *CHI '04 extended abstracts on Human factors in computing systems*, CHI '04, pages 1525–1528, New York, NY, USA, 2004. ACM.

[28] Windows development (Windows). Retrieved December 5, 2010, from `http://msdn.microsoft.com/en-us/library/ee663300%28v=VS.85%29.aspx`.

[29] Windows Media Player. Windows Media Player - Microsoft Windows. Retrieved December 19, 2010, from `http://windows.microsoft.com/en-US/windows/products/windows-media-player`.

[30] A. Worden, N. Walker, K. Bharat, and S. Hudson. Making computers easier for older adults to use: area cursors and sticky icons. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '97, pages 266–271, New York, NY, USA, 1997. ACM.