Univ Access Inf Soc
DOI 10.1007/s10209-013-0295-2

LONG PAPER

Adaptive sliding menubars make existing software more accessible to people with severe motion impairments

Christopher W. Kwan · Isaac Paquette · John J. Magee · Margrit Betke

Abstract The graphical user interfaces of popular software are often inaccessible to people with severe motion impairments, who cannot use the traditional keyboard and mouse, and require an alternative input device. Reaching for buttons and selecting menu items, in particular, can be difficult for nonverbal individuals with quadriplegia, who control the mouse-pointer with head motion via a mouse-replacement system. This paper proposes interaction techniques that can be used with mouse-replacement systems and enable the creation of accessible graphical user interfaces. To illustrate these techniques, the paper presents an image editing application, named Camera Canvas, that uses a sliding toolbar as its universal menu controller. The parameters of the toolbar automatically adapt to the movement abilities of the specific user. Individuals with and without disabilities and of a variety of ages were observed using Camera Canvas. It was found that the developed techniques worked across many different movement abilities and experience levels. Then, it was investigated how such techniques could be used to "retrofit" existing Windows applications with new graphical user interfaces. A tool called Menu Controller was created that can automatically re-render the menus of some existing applications into adaptive sliding toolbars. Menu Controller enables users of mouse-replacement systems to select menu entries that were otherwise inaccessible to them.

1 Introduction

Worldwide, millions of individuals are affected by disorders or injuries that cause severe motion impairments [1]. Their extreme motor impairments may have resulted from traffic accidents, battlefield injuries, brainstem strokes, cerebral palsy, and degenerative neurological diseases, such as muscular dystrophy (MD), multiple sclerosis (MS), or amyotrophic lateral sclerosis (ALS). Individuals who cannot speak and cannot use their hands to operate a computer mouse are extremely limited in their means of communication. Mouse-replacement systems and customized assistive software can immensely improve their daily lives by enabling them to control a computer and thereby communicate with family and caregivers [1, 2]. To fully participate in the information society, however, they need universal access to standard software products [3]. Unfortunately, the graphical user interfaces (GUIs) of popular applications are often inaccessible to people with quadriplegia or other severe motion impairments. This paper describes an adaptive interface solution that enables users of mouse-replacement systems to access GUI buttons and menu items that were otherwise inaccessible or difficult to reach and select. The proposed method is implemented as a "sliding toolbar" that serves as a universal menu controller. The interface is able to first adapt itself to the abilities of the user by measuring the user's actions during simple games. It then allows the user himself or herself to adjust its layout while using it, to better suit the user's specific needs. If some buttons are out of the user's reach, the user can use the sliding toolbar to change the layout on the fly, bringing those buttons nearer to his or her convenient working area.

The design, implementation, and testing of the proposed interface solution were conducted in two phases. In the first phase, a specific application, called Camera Canvas, was developed to investigate the sliding toolbar method. Camera Canvas empowers individuals with motor impairments by enabling them to manipulate photographs, create

C. W. Kwan · I. Paquette · J. J. Magee · M. Betke (✉)
Department of Computer Science,
Boston University, Boston, MA 02215, USA
e-mail: betke@cs.bu.edu
URL: http://www.cs.bu.edu/faculty/betke

Published online: 15 March 2013

drawings, and use it as a canvas for communication and expression. Image editing involves many different mouse interactions, which had to be reimagined for use with a mouse-replacement system. The development of Camera Canvas offered an important opportunity for experimenting with different types of user interfaces and interaction techniques. While the core contribution is the sliding customizable toolbar method, experimentation was also carried out with strategies to simulate clicking-and-dragging and clicking-and holding interactions, provide visual feedback, and reduce accidental selection commands.

In the second phase of the work, after user studies with Camera Canvas helped us to refine the techniques, the functionality of the sliding toolbar method was generalized and the "framework application" Menu Controller was created. Menu Controller automatically retrofits the GUIs of existing Windows applications with adaptive sliding menubars with buttons that enable users of mouse-replacement systems to access the menu entries of these applications.

Preliminary versions of Camera Canvas and Menu Controller have been described in conference proceedings and technical reports [4–6]. This paper describes the most recent versions of the software and explains how several key issues have been addressed that were unsolved in previous versions. In particular, the difficulties experienced by an individual with quadriplegia during a user study prompted further exploration and changes of Menu Controller. Finally, additional user studies involving participants with cerebral palsy are presented.

2 Related work

Users with motor impairments often have difficulties controlling the mouse pointer, for example, keeping it steady while navigating, moving it in desired directions on the screen, and targeting a button or menu item without slipping off or overshooting [3]. Difficulties also include operation of the mouse buttons. Similar to how this work addresses the difficulties that users of mouse-replacement systems have in selecting small, closely grouped menu entries, Worden et al. [7] addressed 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*—mouse pointers with larger than normal activation areas and *sticky icons*—icons that automatically reduce the gain ratio of the mouse pointer when it is on them, making it easier for the mouse pointer to stop or "stick" on the icon. The *Bubble Cursor* [8] 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. [9] also addressed the problem of making user interface targets easier to select. They used 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*. The *ceCursor* by Porta et al. has been designed as a contextual gaze-controlled mouse pointer for use with eye-tracking interfaces. The pointer is represented as a square with four direction buttons placed around it. The user can move the pointer in a stepwise manner, icon-by-icon, or continuously by gazing at a direction button.

The proposed concept of a "sliding menu bar" relates to the idea of reducing the size of an onscreen keyboard by enabling the user to slide into view only those rows of the keyboard that are needed [10]. Spakov and Majaranta [10] showed that text can be entered via such keyboards by gaze.

The work reported here relates to projects on creating framework applications that can provide access to or augment existing software. Akram et al. [11] developed an application mediator to give users of mouse-replacement systems 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 [12]. 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. [13] 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.

The realization that computer interfaces should adapt to the user instead of the user having to adapt to the interface is important [14]. Traditional interfaces are inflexible or at best difficult to customize. Interfaces for users with disabilities should be designed so that they can adapt and be easily modified to cater to the capabilities of the user [15]. Moreover, users should be able to access assistive technology independently, with minimal assistance from caregivers [16]. Recent efforts have focused on creating customizable [17–19] and automatically generated [20] user interfaces for people with motor impairments. Some

users have better control of their movements along certain axes [21], including eye movements [22], some users can only click buttons of a certain minimum size, and some users experience degradation of movement abilities over time.

SUPPLE is a system that automatically generates personalized user interfaces for individual users based on their motor capabilities [20]). The Hierarchical Adaptive Interface Layout (HAIL) model presents specifications for the design of user interfaces that can change and adapt to users with severe motion impairments [17]. 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, a different but related problem was addressed, namely of generating user interfaces for software that already exists and whose source code is not available to be modified. The problem at stake is how to transform these already implemented interfaces to make them more usable and customizable to the needs of users with severe motor impairments.

The work presented here 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* [23], which can simultaneously duplicate dialog boxes across multiple monitors, allowing a user to interact with the dialog in any location, and WinCuts [24], which allows a user to replicate portions of existing windows and interact with them as new independent windows. Stuerzlinger et al. [25] 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 facilities to create new user interfaces using duplicated screen regions, add holes to user interfaces in order to overlay applications on top of one another, and most relevantly modify the interaction behavior of existing user interface widgets or replace them entirely with new ones.

There are also projects that achieve redirection and reverse engineering of user interfaces with image processing. The *SegMan* system [26] 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. [27] improved upon the Microsoft Active Accessibility API's [28] 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* [29] 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.

Camera Canvas is unique as an application for people with severe motion impairments in that it combines photo-editing and drawing functionality. Drawing programs have been popular applications for users of mouse-replacement systems, especially children [1]. A number of drawing programs for users with severe physical disabilities exist. Eagle Paint [1] is a program designed for use with a mouse-replacement system that allows users to draw freeform lines, EyeDraw [30] is a drawing program designed for use with an infrared eye tracker, and VoiceDraw [31] is a drawing program that allows users to draw freeform lines by making different sounds with their voices.

Individuals with extreme paralysis have a choice among various commercial, open-source, and freeware mouse-replacement systems [1, 3] that may be controlled by head motions [32–35] or eye motions [22, 36–41]. User studies were conducted by the authors of this paper with the Camera Mouse interface [32], a popular, freely available video-based mouse-replacement system that enables a user to control the mouse pointer by moving his or her head in front of a camera. Between June 2007 and 2013, Camera Mouse has been downloaded more than 1,000,000 times and is used in schools, hospitals, and private homes worldwide [42]. The interface issues a click command when the mouse pointer has dwelled over a GUI item for a certain amount of time (the default setting is 1 s). Mouse pointer movements can be set to be smoothed, a feature helpful for users with tremor. Various kinds of customized application programs have specially been designed for use with Camera Mouse [1], including text-entry programs, web browsers, games, and drawing programs. In the user studies described in this paper, Eagle Aliens, a simple "shoot the aliens" game, and Eagle Paint, the above-mentioned drawing program, were used. These applications had been specifically developed for Camera Mouse and are freely available on its website [42].

3 Development and evaluation of camera canvas

This section first describes the proposed adaptive sliding toolbar method and how users can control menu items of Camera Canvas with it. It then explains how the complex mouse actions have been redesigned, such as clicking-and-dragging, needed for photo-editing and picture drawing, so that the Camera Canvas software becomes accessible to users of mouse-replacement input systems. The section also reports on studies with users with and without disabilities, who used Camera Canvas with the mouse-replacement system Camera Mouse.

3.1 Sliding toolbar

The main user interface element of Camera Canvas is the Sliding Toolbar (Fig. 1 top). It consists of two panels: a tool menu panel containing specific image editing tools and a navigation panel containing navigation buttons. The user can reposition the tool panel by sliding it sideways using the Prev and Next buttons in the navigation panel (Fig. 1). This sliding ability addresses the problem of some users only having good movement control within a certain range of the center of the screen. If users cannot reach a tool button on the edge of the screen, they can select Prev or Next to slide the tool buttons toward the center. The direction of movement is from the perspective of the button currently in the center position of the toolbar (in Fig. 1 top, the Zoom button, in Fig. 1 bottom, the Color Choice button). Pressing the Prev button will cause the button in the previous position to the center position to slide to the center position. Similarly, pressing the Next button will cause the button in the next position after the center position to slide to the center position. As long as the user keeps the mouse pointer on top of the Prev or Next button, the toolbar will continue to automatically slide on an adjustable interval.

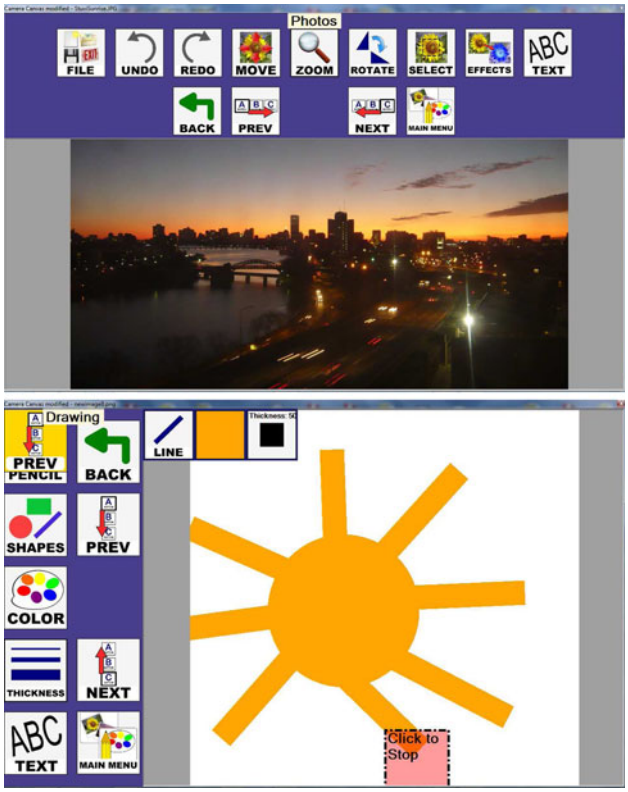

Fig. 1 *Top* Camera Canvas in Photo-editing mode with a *horizontal-top* layout and small buttons. *Bottom* Drawing mode with a *vertical-left* layout and larger buttons (*right*). The second set of Prev and Next buttons signify that the toolbar includes more buttons off-screen

When a tool from the tool menu panel is selected, if that tool has a submenu, the buttons of that submenu will replace the current buttons in the tool menu. The user can get back to the previous menu of buttons by pressing the Back button in the navigation panel or go back to the top-most tool menu by clicking the Main Menu button in the navigation panel.

Camera Canvas has three configurable settings in the Settings menu: toolbar placement, button size, and toolbar sliding speed. These settings can all be changed at run-time using tools within the application. The tools are designed to be easy to use so that the user can actually modify the configuration himself. The placement and orientation of the toolbar can be changed to four settings: Horizontal-Top (Fig. 1, left), Horizontal-Bottom, Vertical-Left (Fig. 1, right), and Vertical-Right.

Each setting aims to constrain movement primarily along a single axis and in a single area of the screen to address the challenges of users having better movement abilities along different axes and users being able to reach different areas of the screen more easily than other areas.

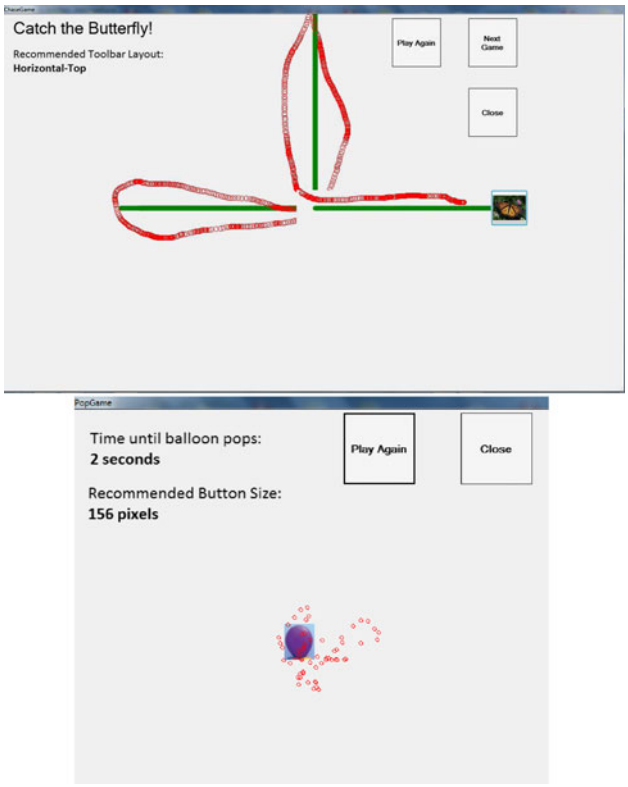

Fig. 2 The "Catch the Butterfly" game recommends which axis and area of the screen are best for the user by having her follow a butterfly (*left*). *Green lines* show ideal mouse trajectory, *red circles* show actual trajectory. The "Pop the Balloon" game recommends a button size for the user by having her try to keep the mouse pointer still within a small area (*right*). The *balloon* is the ideal area; *red circles* show the actual mouse movement area

The size of all buttons in the application can be made smaller or larger to address the challenge of different people being able to click buttons of different minimum sizes. Finally, the interval at which the toolbar buttons slide can also be adjusted so that the buttons slide faster or slower. The Settings menu also contains a configuration wizard for Camera Canvas in the form of two simple, easy to understand games (Fig. 2). With these games, a user's performance can be analyzed automatically, so that settings for Camera Canvas can be recommended that would make it the most usable for the specific user.

3.2 Camera canvas: photo-editing tools

For the Photo-editing mode of Camera Canvas, several interaction techniques were developed to make common photo-editing tasks possible with camera-based mouse-replacement systems. The Move and Zoom tools place four translucent arrows in the middle of the screen. To pan around the image, the user puts the mouse pointer over one of the arrows and the image automatically moves until the user moves the mouse pointer off of the arrow (Fig. 3, top left and bottom). No matter the size of the image, the user only needs to make small movements between the arrows to pan, instead of having to physically move the mouse pointer around the entire image. The Rotate tool uses a custom user interface component called a Choice Form (Fig. 3, top right) that is an alternative to components such as sliders or small increment arrows, which are challenging for users who have difficulties controlling the mouse pointer. The middle of the Choice Form contains a preview of the rotated image so that the user can see the effects of the rotation before actually committing the change. The Choice Form is also used by many other tools in Camera Canvas.

Instead of the traditional click-and-drag method of selecting a portion of an image, the Select tool uses two sets of arrows similar to the ones used in the Move and Zoom tools. When using Select, a translucent blue rectangle (representing the selection) and two sets of arrows appear in the center of the image, for the user's convenient access. The set of four arrows on the left control the position of the top left corner of the selection box and the set of four arrows on the right control the position of the bottom-right corner of the selection box. By moving the mouse pointer into these arrows, the user can control the

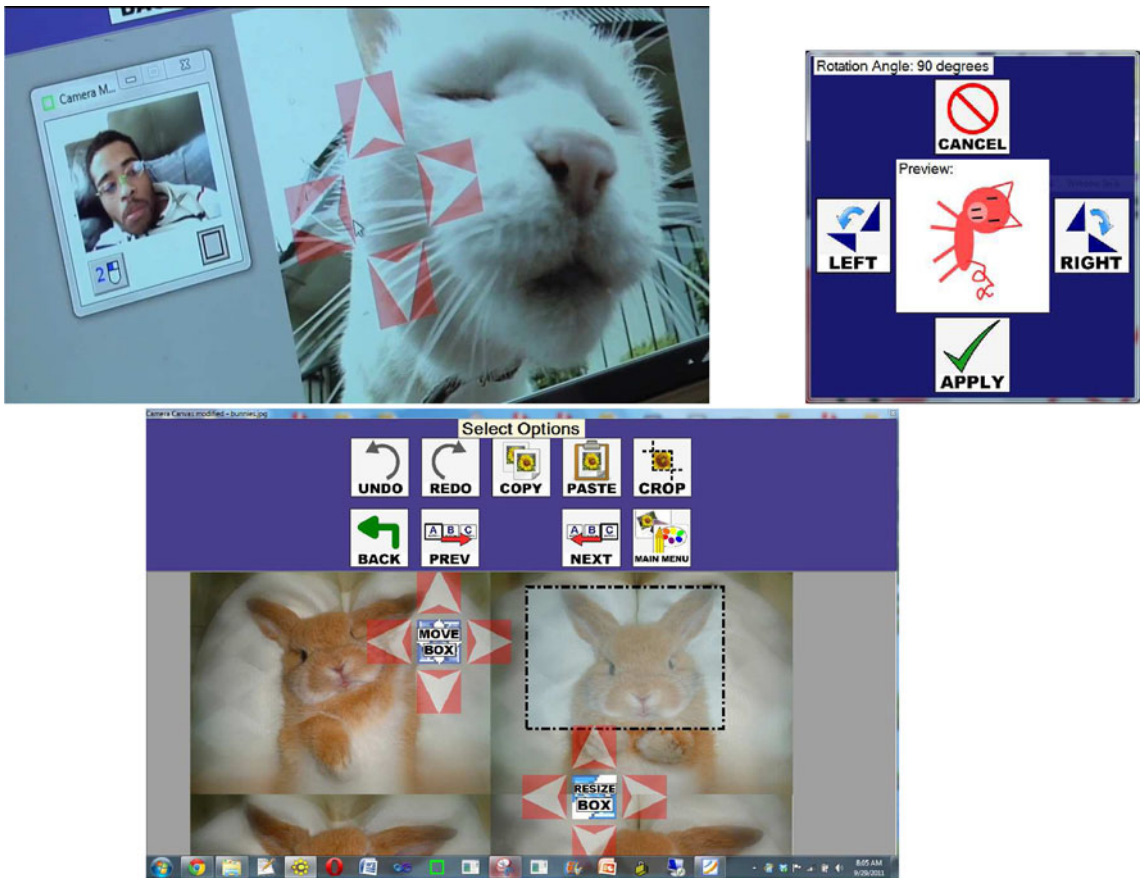

Fig. 3 Photo-editing with Camera Canvas. *Top left*: Camera mouse user with Moving Tool. *Top right*: Rotate Tool. *Bottom*: Select Tool

position and size of the selection box. The two sets of arrows never change positions from the center of the image, so no matter the size of the selection, the user can control it using only small movements between the two sets of arrows. Once the selection box is of the desired position and size, the user can then cut, copy, paste, or crop the selection.

3.3 Camera canvas: drawing tools

The Camera Canvas interaction for drawing straight lines and geometric shapes was inspired by the drawing process in EyeDraw [30]. To address the "Midas touch" problem [43] for drawing (how to differentiate looking at the picture versus actually drawing the picture), the researchers of EyeDraw created a system where if the user looked at one spot for some amount of time, the cursor would change colors to signify that drawing was about to begin; if the user was just looking and did not want to actually start drawing, they would just need to look elsewhere.

In Camera Canvas, to start drawing, the user must first dwell on the area where she would like to place the starting point of her drawing. After a click is registered, a green helper box appears where she clicked to signal that drawing is about to begin. If the user would actually like to start drawing at that point, she keeps the mouse pointer in the green Helper Box long enough for another click to register and then drawing begins. If the user does not want to place the starting point at that location, she only needs to move the mouse pointer out of the green Helper Box and it disappears, resetting the process. As the user is drawing, the line or shape is continuously redrawn with the ending point at the current position of the cursor. When the user wants to end the drawing, she dwells where she would like to end the drawing and a red Helper Box appears. If she would in fact like to place the end point of the drawing at that point, she just needs to keep the pointer inside of the red Helper Box. If she does not want to place the end point there and instead wants to continue drawing, she just needs to move the mouse pointer out of the red Helper Box and it disappears. The sizes of the Helper Boxes are the same size as the toolbar buttons and will change if the button size is changed. The drawing process is outlined in Fig. 4, top.

Instead of using a traditional color palette which relies on sliders or clicking of a precise point in a color wheel, a simple color palette was implemented that is much more usable with Camera Mouse, but still gives users a fair amount of color variety. The color menu (Fig. 4, bottom) first displays a set of primary colors: black, white, brown, red, orange, yellow, green, blue, and violet. When the user clicks on a primary color, nine different shades of that color are then automatically generated for the user to choose from.

3.4 Camera canvas experiments, results, and conclusions

Several user studies were conducted to obtain a qualitative assessment of the program use.

3.4.1 Methodology of experiments and technical details

A total of 28 users without and 3 users with physical disabilities participated in the study, with ages ranging from elementary, middle, and high school age, college age and middle age. The users without disabilities had never used the Camera Mouse input system before. The participants were asked to use various Photo-editing tools to manipulate a photograph and various Drawing tools to draw a shape. They were then asked to play around with changing different configuration settings. There was no strict test plan; users were given freedom to explore the different features of the program as the researchers observed them. The studies were conducted in the summer and fall of 2010 and involved several multi-hour computer sessions.

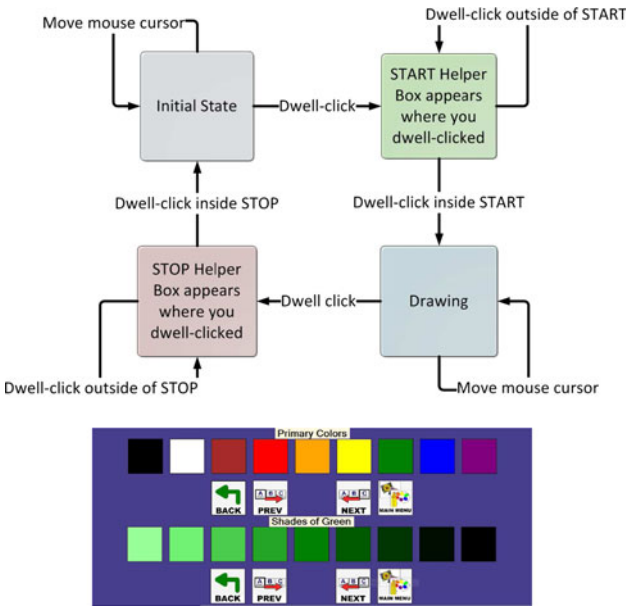

Fig. 4 *Top*: The drawing process in Camera Canvas. *Bottom*: The Camera Canvas *color palette* generating different *shades of green*

Fig. 5 Drawings created by users without disabilities

Fig. 6 *Left*: A user with cerebral palsy (User C) interacting with Camera Canvas using the Camera Mouse. *Right*: An image edited by a user with severe cerebral palsy (User R). He was able to rotate the image (presented to him *upside-down*) and experiment with drawing several shapes on the image

Camera Canvas was designed for Windows and can handle various versions of Windows-based operating systems. The evaluation studies were conducted with laptop and desktop computers by Sony, Alienware, and Dell with Intel dual or quad core CPU speeds ranging from 1.83 to 2.83 GHz and running the 32-bit versions of Windows 7 Professional, Windows Vista, and Windows Server 2003, respectively. Computer screen resolution was 1280 × 800, 1920 × 1,200, and 1,280 × 1024, respectively. The laptops computers had built-in cameras located above the laptop screens with resolutions 640 × 480 (Sony) and 1280 × 960 (Alienware), respectively. The desktop Dell computers were used with Logitech QuickCam Orbit cameras that were placed on the desk in front of the users. The distance between users and cameras ranged from 35 to 60 cm. The computers were located in typical office environments with fluorescent lights from above and people having conversations and moving around in the background. An exception was the environment of user study 2, which was conducted in the participant's home. At first, it was too dark in his living room for Camera Mouse to track well, so a lamp was turned on and pointed toward user C. The positioning of C was also unusual. User C has to sit in a wheelchair all day at school, so when at home, he is more comfortable sitting in a deeply reclined position. The laptop was placed on a tray located above C's abdomen. C's head was ca. 50 cm from the screen (Fig. 6, left).

3.4.2 Studies with users without physical impairments

The participants in the experiments found the software easy to understand and use even without prior experience using Camera Mouse. With a little experimentation time, users without disabilities were quickly able to start drawing shapes and manipulating images. It was found that nearly all users enjoyed the drawing tools the most and spent most of their time with the program drawing (Fig. 5). The users provided valuable feedback on which features needed improvement, and also what features they wanted to see in future versions. Common suggestions were a fill tool and clip-art stamps.

3.4.3 User study 1

A user study was conducted with G, a 13-year-old student in the 6th grade who has cerebral palsy. G had never used Camera Mouse before, but was eager to try out the software. The researchers first introduced the Camera Mouse to her, then she was asked to try out moving the mouse pointer by moving her head, and it was shown to her how to play with Eagle Aliens and Eagle Paint. G then wanted to try out Camera Canvas. She first received an introduction of the Camera Canvas functionality and interface components. In particular, a quick overview of the sliding toolbar user interface element was presented, explaining how it works and how its settings can be adjusted. When using Camera Mouse, G had trouble keeping the mouse pointer still in small areas long enough for a click to register. Even after an attempt had been made in adjusting the Camera Mouse settings for dwell time, it was still difficult for G to click with the Camera Mouse. G also was having difficulties because buttons on the Camera Canvas toolbar were too close together, and therefore, neighboring buttons were easy to click on by accident. G has some control of her index finger and can use it to operate the touch pad of the laptop. When G had trouble using Camera Mouse, she would use her hand with the touch pad to select options instead. Since G was having difficulties with Camera Mouse, the evaluation of Camera Canvas was continued with her using the touch pad. After some practice, G understood how the sliding toolbar worked and was able to select different options. She was able to open a pre-loaded image of a cat, select a shade of purple, and draw lines on the image.

3.4.4 User study 2

Another participant was C, a 16-year-old high school student with cerebral palsy (Fig. 6 left). C primarily interacts with his computer using Dragon voice recognition software [44] and also has some control in his index finger which allows him to use the touch pad on the laptop. He had never used camera-based assistive technology before, so in the

first session, he was given an introduction to Camera Mouse and some of the software developed for it. C quickly understood how to use Camera Mouse. He enjoyed adjusting the settings of Camera Mouse himself. C tried to do most things by moving the mouse pointer with Camera Mouse, but also used his finger with the touch pad if actions were too difficult with Camera Mouse. C was first asked to play the game Eagle and then he moved on to Camera Canvas. C was able to understand and use the sliding toolbar and several of the tools. He was very inquisitive and liked to experiment, drawing freeform lines, something that he had never done before. Once he became more familiar with the freeform drawing tool, he was able to open an image of his family from his computer and use the tool to draw hair on top of one his family member's heads.

3.4.5 User study 3

The user studies also involved a non-verbal adult, R, with severe cerebral palsy and quadriplegia (Fig. 6 right). His level of cognitive function is very high, but his movement abilities are extremely limited. He cannot control his index finger like users G and C, and was completely dependent on Camera Mouse to move the mouse pointer. User R had participated in experiments with the initial version of Camera Canvas [4]. In the prior experiments, R was excited about the prospect of manipulating images but was unable to use the majority of the features [1]. In the experiments with the current version, the general purpose of the program was explained to him, how the toolbar worked, and its customization possibilities regarding position, button size, and sliding speed. R had a difficult time reaching buttons at the top of the screen, so a Camera Canvas configuration with large buttons in a Horizontal-Bottom layout was selected.

User R understood how the Camera Canvas interface worked. Initially, it was difficult for him to keep the mouse pointer on top of one button long enough for the click to register. Shortening the time required for a dwell-time click in the Camera Mouse settings helped reduce the problem but it still persisted. R was able to use the Prev and Next buttons to slide the toolbar buttons he wanted toward the middle of the screen. Using the Drawing mode, he was able to select different shapes and then draw rectangles around the image (Fig. 6, right). It is not known whether R intended to draw something specific or was just experimenting with the tool, as this may have been the first time that R interacted with a drawing interface. Using the Photo-editing mode, R was able to successfully use the Move and Zoom features to zoom the image to a greater magnification and then pan the image so that a particular portion was centered on the screen. R was also able to apply the Invert Colors feature to the image and then undo the change.

The user study with R revealed a number of challenges. R could slide to buttons that he wanted to reach, but oftentimes he would slide the toolbar too much and overshoot the button he wanted or would accidentally activate the Prev or Next button when trying to select a button in the tool menu, causing his intended target to shift. To address this problem, the researcher tried to slow down the sliding speed setting, but R still hit the Prev and Next buttons by accident because of their proximity to the tool menu buttons. R also accidentally selected buttons next to his intended buttons. A particularly frustrating experience for him was accidentally hitting the Main Menu button when he was in the middle of trying to apply an effect to the image. Hitting the button by accident would take R all the way to the Main Menu of the program and then he would have to click on Photos, slide down to effects, and then click on the effect again. This happened multiple times and eventually the researcher took control of the mouse in order to get him back to the Effects menu again.

The observation of accidental activation of buttons suggests that the buttons should be spaced farther apart or that this setting should also be adjustable. The observations that R had to keep sliding to reach buttons near the edge of the screen and that he kept hitting buttons accidentally suggest that he might benefit from the toolbar having fewer buttons. A greater number of buttons on the toolbar increases the chance for error. It may also be cognitively overwhelming for someone using the program for the first time. Perhaps a more hierarchical approach (more levels with fewer buttons at each level) would be more usable for this user. Even though the arrows of the Move feature were a fixed size that may have been too small for R, he was able to select them. This was because the Move command was activated whenever the mouse entered the arrow region, rather than forcing the user to hold the mouse pointer in the region for a specific duration, as is the case with buttons. This suggests that a boundary-crossing or mouse-touch approach instead of dwell-activated buttons might be more usable for R.

The ability to configure the user interface of Camera Canvas was very important in the experiments in which R participated. All three of the configuration options (toolbar placement, button size, and sliding speed) were used to try to provide the most usable layout for the user. R also played the configuration games. He was able to understand and complete both the butterfly (toolbar layout configuration) and the balloon (button size configuration) games, although the layouts of the buttons in both games could be improved or ideally made configurable. The automated recommendation system proposed a Vertical-Right layout with buttons of size 160 $\times$ 160 pixels for the user. The user was satisfied with these settings and chose to keep them for the remainder of the experiment. Although the user liked these settings, it is not known whether there were settings

that could have made the program even easier for him to use because trying different settings was stopped after the user indicated he was satisfied.

3.4.6 Conclusions

A great deal was learned from the user studies with G, C, and R. Problems with the interface were seen that did not arise when testing the software with users without disabilities. Teenagers G and C enthusiastically embraced Camera Canvas as a new canvas to express themselves. While there were many features that R had trouble with, could not use, or did not try, in general, the experiments revealed a major improvement over the experiences R had with the initial version of the software.

4 Development and evaluation of menu controller

Menu Controller was developed to retrofit existing applications with new graphical user interfaces in order to make them more accessible for people with severe motor impairments. 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 the need 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 motor impairments. The main initial challenge in developing Menu Controller was to find a method for harvesting menu items. Then, an appropriate way for displaying the harvested menu entries was explored. An approach was chosen based on the two-part sliding toolbar that had been developed for Camera Canvas.

The recommendations from an initial user study with Menu Controller [6] let to an update of Menu Controller to include additional features, namely the possibility to control the menus of additional Windows applications, support for customizing the size of buttons, the possibility to control the display behavior of the toolbar, and finally the possibility for users to customize the location of the toolbar.

4.1 Menu controller: re-rendering the user interface of applications

Once Menu Controller has gained access to the menu entries of an application (see next section), it can re-render them in a way that enables users with limited motion control to select them. A re-rendering approach was developed that was based on the sliding toolbar design of Camera Canvas [5]. The toolbar contains buttons that represent the menu entries of the original interface of the application.

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 (Fig. 7 top). When a user clicks on an entry in the root menu of the application, a submenu is typically displayed (Fig. 7 bottom). The same behavior is achieved in Menu Controller. When one of the 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.

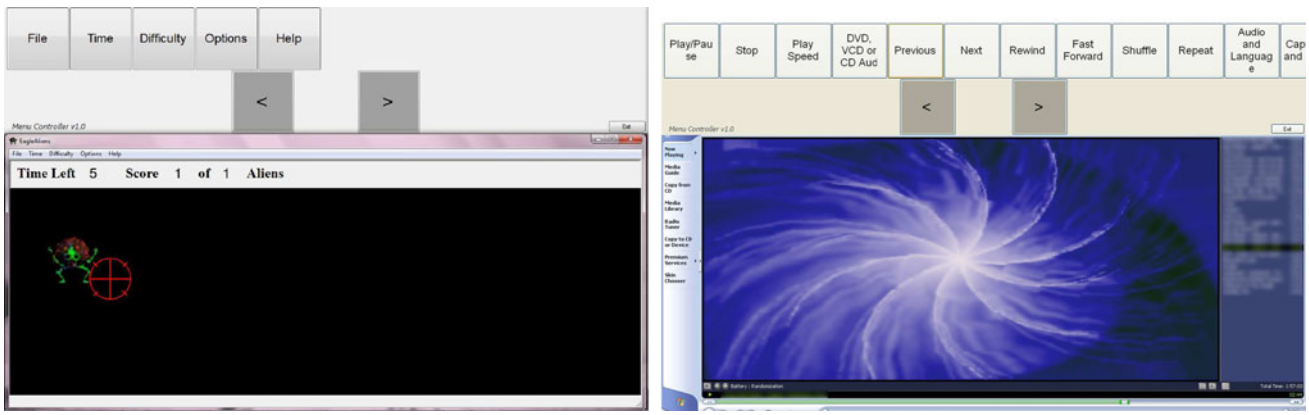

Fig. 7 *Top*: Menu Controller re-rendering of top level menu of Eagle Aliens game [1] designed for Camera Mouse [32]. *Bottom*: Menu Controller re-rendering of Play submenu in Windows Media Player 9 [45]

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. As for Camera Canvas, 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 toward the middle of the screen, until the button is within the user's reach.

4.2 Menu controller—harvesting menus of applications

It was decided to develop Menu Controller for the Microsoft Windows operating systems because many of the applications that users of mouse-replacement systems desire to access run-on Windows. The initial version of Menu Controller [6], written in C# code, can manipulate the menu items of such Windows applications using the Windows API [46]. The Windows API allows a program 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 the 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 an external program the power to control almost any aspect of any window without knowledge of the inner workings of the window itself. In the current scenario, Menu Controller is the external program that controls the application that a user with quadriplegia wants to access, for example, the Windows calculator. To control the View submenu of the Windows calculator (Fig. 8), the Menu Controller does not need to access the calculator code itself.

While working on the first version of Menu Controller [6], it was found that the MenuAPI [48] was unable to control all menus. Specifically, it was assumed that there were two types of menus: menus accessible via the MenuAPI and menus accessible through a Microsoft technology called Active Accessibility [28]. It was then found that Active Accessibility has since been supplanted by a newer .NET-supported technology called UI Automation [49], which according to Microsoft, "offers many improvements over Active Accessibility." The researchers therefore decided to use UI Automation to support the menus that they could not access through the MenuAPI, instead of using Active Accessibility [6].

When Menu Controller is run, a timer is started. When receiving a once-a-second "tick" event from the timer, Menu Controller determines which window currently has the focus. If no window currently has the 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 retrieves a handle that points specifically to the menu of that window, reads the first level of the menu, i.e., the part of the menu visible to the user prior to clicking on any menu items, and stores information about each first-level menu entry 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. (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 determine whether 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.

Before any of the existing code was refactored to incorporate UI Automation support, it was necessary to ensure that the required changes would minimize code divergence. In particular, the toolbar-building code should not take two different paths: one for MenuAPI-enabled windows and another for UI Automation-enabled windows. The first step was to move all menu logic code into a separate class, making the code that controls user interaction and what the user sees in Menu Controller almost entirely menu-type agnostic. This should make future updates to the software easier.

Following are the details of how Menu Controller takes slightly different paths for MenuAPI-enabled versus UI Automation-enabled applications. First, Menu Controller checks whether the in-focus window supports the MenuAPI. If it does, the code behaves the way it did in the previous version of Menu Controller with respect to how the menu entries are harvested. If entries are found, a collection of "MenuItem objects" is created and stored. If Menu Controller does not find that the window is MenuAPI-enabled, a new code-path attempts to retrieve the

first-level menu using UI Automation. If this comes back with menu entries, a similar list of objects is created, using the generic UI Automation object that stores information about each menu entry. The advantage of this approach is that the code that controls all the toolbar logic, only has to take different paths at two places, namely in the generic click method that is invoked whenever a button on Menu Controller is clicked, and when it is necessary to identify the type of button that is being added to the toolbar. All other code remains identical for both approaches. Again, this decision should help in future efforts to maintain or update the code-base.

4.3 Menu controller: initial user study

The initial Menu Controller user study included User R, the non-verbal adult with quadriplegia (Fig. 6). The same experimental setup as described in Sect. 3.4 was used. User R was asked to play with Eagle Aliens, a game where the user moves the mouse pointer around the screen to "shoot aliens" (Fig. 7 left), and Eagle Paint, a freeform line drawing program. Both are popular programs designed for use with Camera Mouse [1] that the participant was already familiar with. It was explained to User R that the goal of Menu Controller to allow a user to operate more of the features of the program by himself. The participant especially liked playing Eagle Aliens and seemed excited at the prospect of starting a new game or adjusting 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, R 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. It was difficult for R to reach buttons that Menu Controller displayed in the top left corner of the screen. The researcher explained to R 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 R. He 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, R 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. It became clear 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, was a much needed change.

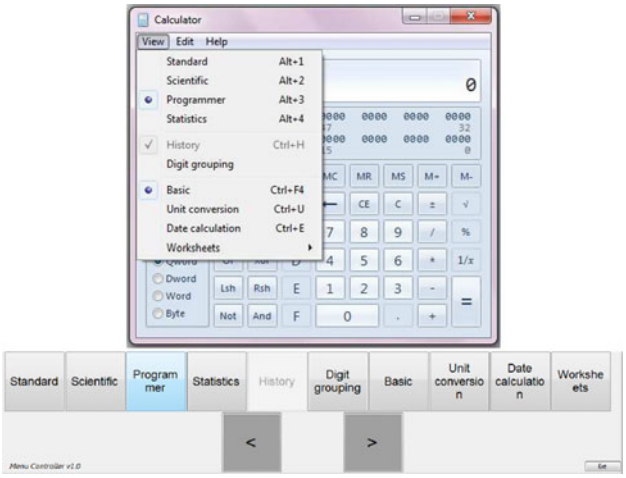

Fig. 8 *Top*: The View submenu of the Microsoft Windows 7 Calculator [47] program. The menu entries are small and closely grouped together, making them difficult to access using a video-based mouse-replacement system. *Bottom*: Automatic re-rendering of the View submenu by Menu Controller

Although User R was able to hit some of the buttons, in general, it was difficult for him to make movements to reach the top of the screen. To try to help R, the researcher re-initialized Camera Mouse and adjusted the mouse movement gain settings, but R still had difficulties. It would be beneficial for R 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 R tried to click on a button he would often click neighboring buttons by mistake. It would be very helpful for him to be able to adjust the button size and space between buttons at runtime.

When User R 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," presented R with a list of options, only one of which could 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 (Fig. 8). However, in the version of Menu Controller used here, there was not an equivalent means to provide this type of 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, R was 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.

A lot was learned from R's participation in the user experiment, his interaction with Menu Controller and the two application programs. The areas where improvements were necessary were identified. It was very encouraging to see R's positive reaction to the software despite the difficulties he had in using it, providing further motivation to develop a new version of the program as usable as possible.

4.4 Menu Controller: changes to improve usability

From the first user study, it was apparent that the one-size-fits-all toolbar the initial version of Menu Controller displayed was sufficiently accessible to some users with severe motor impairments. The program re-rendered the toolbar at the top of the screen with buttons that did not have spacing between them and hard-coded button size. The next priority was therefore to provide more options for the way Menu Controller displays its toolbar and buttons. Three areas that needed customization were identified: (1) button sizes and spacing between buttons should be customizable; (2) because button size would be customizable, a different display behavior for the toolbar was needed, e.g., the toolbar should not take up more and more room as the button size increased; and (3) the toolbar should have the ability to be placed at different screen locations depending on user preference. The first step in making the above options customizable was to create a settings page for Menu Controller, which gives users the ability to adjust the button size, choose the display behavior of Menu Controller, and to customize the location of the toolbar. The benefits of each of these customizations are discussed below.

4.4.1 Button size customization

Having the ability to adjust the button size allows users with less severe motor impairments to choose smaller buttons that do not take up as much "screen real estate." However, those who so desire can make the buttons much larger, making them more accessible, but at the cost of losing more screen real estate (a solution to this problem is explained below).

4.4.2 Auto-hiding the toolbar

Giving the user more options for the behavior of the toolbar became critical, after the ability to make buttons larger had been provided. A key breakthrough to compensate for larger buttons was the idea of having a collapsible (or auto-hiding) toolbar that only appears in full when a user requires it. If the user chooses the auto-hide option from the settings page, the toolbar is displayed at the top of the screen as a thin strip. In addition, there is now an indicator on the toolbar that tells the user whether or not the active window is supported by Menu Controller (Fig. 9, left). This feature provides a visual indicator to the user that she need not try to access Menu Controller in cases where the in-focus window is not supported. If the window is supported, however, the user can move the mouse cursor on top of Menu Controller's thin strip, causing Menu Controller (with all the buttons, etc.) to appear (Fig. 9, right). This feature allows the user to make the buttons as large as needed without losing valuable "screen real estate."

4.4.3 Automatically resizing program windows

Another feature available from the "display behavior" dropdown of the settings page is the ability to have windows auto-resize in order to fit in the area directly below the toolbar. This feature is especially useful for users who either have a large

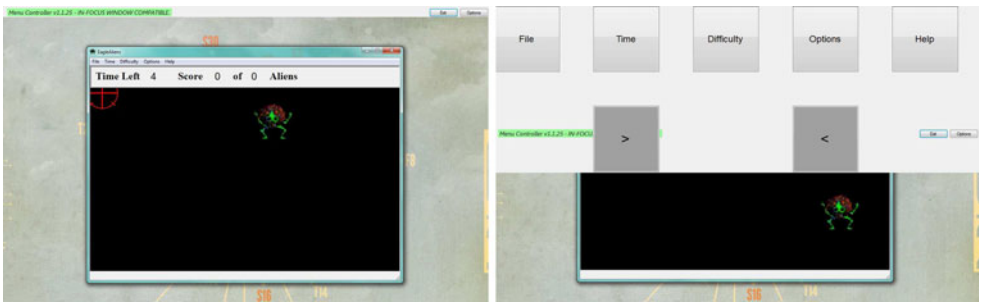

Fig. 9 *Left*: Display behavior set to "Auto-hide Menu Controller" for Eagle Aliens game. The mouse pointer (not shown) is not on Menu Controller, so Menu Controller is in minimized mode at the top of the screen. *Right*: Display behavior set to "Auto-hide Menu Controller." The pointer (not shown) has moved on top of Menu Controller, so Menu Controller becomes activated. Once the pointer moves off of Menu Controller, Menu Controller minimizes

monitor (and do not wish to Menu Controller to auto-hide its toolbar) or choose a small-enough button size so that the toolbar does not take up too much room on the screen. With the first version, it was noticed that sometimes windows would appear either under or over the toolbar. This could become a nuisance to users who wish to either reach an area of the current program covered by Menu Controller or conversely users who are unable to reach Menu Controller because the in-focus window is covering the toolbar. By resizing the window to take up the area directly below Menu Controller (Fig. 10 left), the above issues are resolved. An additional benefit is that the user is now less likely to accidentally click off of the window with which he is working. In essence, this feature provides a "maximize" option for the in-focus window within the confines of the toolbar of Menu Controller.

4.4.4 Customizing toolbar location

The ability to place the toolbar in other locations on the screen was something the researchers deemed necessary after conducting the first user study. The user found it difficult to lift his head to make the mouse go to the top of the screen. For this reason, it was decided to make the location of the toolbar customizable, following an approach similar to Camera Canvas. Now, the user can select from the toolbar orientations: Top (default), Left (Fig. 10 right), Right, and Bottom (if Bottom is selected, it is recommended that the Windows taskbar be set to auto-hide).

4.4.5 Additional changes of Menu Controller

Additional noteworthy changes are now discussed that were incorporated into the latest release of Menu Controller.

First, a space was added between the buttons that is equal to half the size of the buttons themselves. This provides the user with "rest areas" that they can place the mouse pointer on without risking undesired clicks to the buttons themselves. The first version of Menu Controller included buttons without sufficient spacing, making it easy for users to unintentionally click buttons while deciding what action they wanted to perform next.

Second, it was decided to swap the left/right arrow buttons to make them work in what is called "curtain style"—i.e., instead of clicking left and right to have the buttons move left and right, respectively, the left/right buttons now draw buttons to the right and left, respectively, which seems to be a more intuitive approach judging from the initial difficulties in understanding the behavior of the toolbar encountered by the user in the first case study. Also, changes were made to make the buttons, when moved from left to right, align themselves with the left/right arrow buttons, providing a "cleaner" more grid-like look to the toolbar.

Third, an issue was discovered when resizing was added: the smaller button sizes caused the text to become illegible. To solve this problem, tooltip now appears when the user places the mouse pointer over a given button. This way, users can have smaller buttons, while still being able to see the button text.

Fourth, another issue to be fixed after the first user study, was the fact that it was unclear to the user whether a button was actually clicked. This is because Camera Mouse [32] simulates a mouse click after a period of inactivity (i.e., mouse movement). Since the user is not actually clicking the mouse directly, the original version of Menu Controller did not make it obvious that a button had actually been clicked. This was especially pronounced when a button did not actually cause an action to occur in the active window, but was a toggle or check-mark menu entry, such as in the case of Eagle Alien's difficulty selector. Initially, it was hoped was that the type of menu entry could be detected (e.g., action versus toggle versus checkmark) and the buttons could be colored differently to show the user whether menu entries were checked or unchecked or indicate which button in a group (i.e., a toggle button) was currently selected. However, this proved to be more difficult than initially thought. The menu entries do not seem to refresh themselves internally when they receive a window message from Menu Controller in the same way they do when they are clicked directly by the user from the menus themselves. To provide a compromise and still give the user an indicator of some type, the last-clicked button highlights,

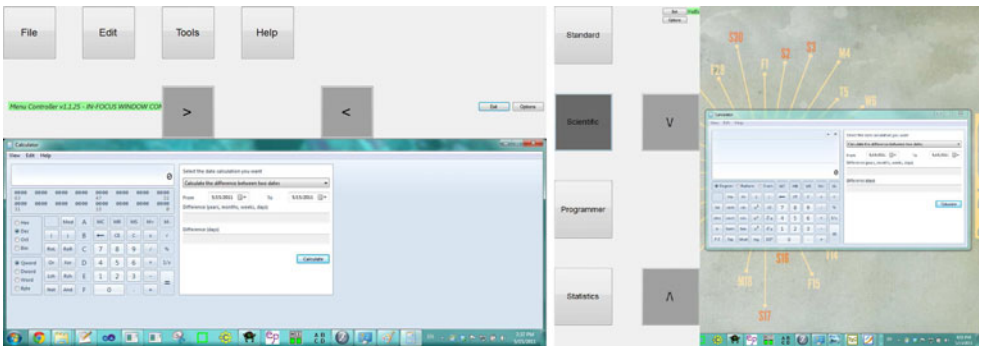

Fig. 10 *Left*: When the display behavior of Menu Controller is set to "Auto-resize compatible windows," Menu Controller automatically resizes Windows Calculator to fill up the remaining space on the screen so that the user cannot click off of the program accidentally. *Right*: The orientation of Menu Controller set to "Vertical-Left" with large buttons of size 175 × 175 pixels

which allows the user to see that something has been done after the click, and avoids having the user click multiple times expecting to see some visible change.

Finally, it is worth noting all settings are stored in the settings page per Windows user. Windows computers can be shared among several users, i.e., one user can log off Windows, allowing another user to log in with a different user name and password. The settings are stored for each log-on, making the settings customizable to suit different user's needs.

4.5 Menu Controller: second user study

The participant of the second user study with Menu Controller is the 16-year-old high school student with cerebral palsy, User C, who had participated in the testing of Camera Canvas. It was suggested to C that it might be beneficial for him to try out Menu Controller, which could potentially allow him to use more programs on his computer through an interface that he is already familiar with from Camera Canvas. In a subsequent session with C, he was introduced to the Menu Controller project. The motivation behind Menu Controller was explained to C, focusing on how small menu entries were difficult to access using the Camera Mouse, and C responded with "I can imagine." It was explained how the Menu Controller interface was similar to the Camera Canvas sliding toolbar, which C was already familiar with. The possibility of adjusting, just like in Camera Canvas, the placement, orientation, and size of the toolbar buttons, was explained, and C thought that the idea was "cool." As C was very curious about how the software works, he was given a high-level explanation of how Menu Controller harvests menu entries and simulates commands.

The Eagle Aliens game was then opened. It was pointed out how Eagle Aliens had a small menu at the top with the options: "File," "Time," "Difficulty," "Options," and "Help" (Fig. 9). Menu Controller was launched in minimized mode where it sits docked as a small strip at the top of the screen and only displays its buttons when the user puts the mouse over it. C was asked to try adjusting the difficulty level of the game using Menu Controller. After it was explained to C that the program was in minimized mode and that he would have to hit the top of the screen for Menu Controller to appear, he responded, "Oh, I got it."

User C was able to reach the top of the screen with the mouse pointer and activate Menu Controller. He said, "I want to try [difficulty level] 'Hard,' " and proceeded to select the "Hard" button that represented these actions using Menu Controller (Fig. 11).

Next C was asked to try Menu Controller with an existing program not designed for use with Camera Mouse. The Windows Calculator program was opened, showing

Fig. 11 *Top*: A user with motor impairments (User C) playing with the Eagle Aliens game using Menu Controller. *Bottom*: A user with quadriplegia (User R) adjusting the difficulty level of the Eagle Aliens game using Menu Controller

him how Menu Controller also grabbed the menu entries of the calculator. C was then asked to try changing the view of the calculator to the scientific view. He was able to again activate Menu Controller by going to the top of the screen, click on the "View" button and then click on the "Scientific" button in the sub-menu of the "View" button (Fig. 10 right).

4.6 Menu controller: limitations

Considerable efforts were expended on trying to make Menu Controller work with the browsers Internet Explorer [50] and Firefox [51]. Although a certain measure of success was achieved with Internet Explorer, in that Menu Controller was enabled to access the first-level menu of Internet Explorer, accessing any submenus or enabling Menu Controller to simulate clicks on any of the menu entries was not possible. These submenus and menu entries do not seem to have been made available to UI Automation. The most pressing remaining issue yet to be resolved is therefore the ability of Menu Controller to work with a web browser.

The current version of Menu Controller does not provide a way for the user to go back to a parent submenu. Once a submenu is displayed, the only way for the user to get pack to the parent menu is to click off of Menu Controller onto

the application window and then locating the parent again by clicking through Menu Controller.

One idea that could be beneficial for Menu Controller would be to add some type of visual indicator to show the user where in the menu hierarchy they are. Currently, the user could be several levels deep and not know where he or she actually is.

5 Discussion

The number of systems that analyze the facial gestures of computer users has been growing explosively [52]. Formal evaluation of such systems with user studies has been criticized as insufficient [53]. Empirical user studies, if conducted at all, generally do not include users with motion disabilities. Mouse-replacement systems that have been tested with people with motion impairments are *Sina* from Spain [34, 54] and *Nouse* from Canada [33]. For other promising research systems [55–60], tests with users with motion impairments have not been conducted but are reportedly planned.

Studies have found that adoption of software to assist computer users with severe motor impairments is sometimes difficult. Dawe [61] 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 %. The goal in designing the systems described in this paper was therefore to make them intuitive and easy to use in order to attract a user community that will find it beneficial. Moreover, it was important to involve individuals in their user studies who belong to the user population for which the software is designed. Users with severe motor impairments who did not have cognitive disabilities were involved. Plans exist for future studies to include users with a limited level of cognitive functioning.

Camera Canvas and Menu Controller will be freely available to the extensive worldwide Camera Mouse user base [62] for download from the Camera Mouse research website [63]. The authors hope to encourage individuals with motion impairments to explore whether they can gain access to applications that they had not been able to use previously.

It is important for the users to have a consistent, customized, and accessible user experience with the applications they try out. The great variety in human–computer interfaces, often presenting difficulties for users without disabilities, bring even more frustration for users with disabilities who may have to customize their input devices for every application [16]. With Menu Controller, the hope is to alleviate some of this frustration by reducing the number of different application interfaces that users will have to customize their input devices against.

Initial user studies with Camera Canvas and Menu Controller helped to identify some of the areas where the first versions of the software needed improvement. Subsequent experiments with new versions showed that the usability of both programs was improved upon by focusing on the button placement and the location of the toolbar, along with behavior customizations to compensate for button size and distance, and, for Menu Controller, increasing the amount of programs it supports. The current versions of Camera Canvas and Menu Controller give users the ability to customize their interfaces to suit some of their needs.

A priority with the additional menu support effort described above was to make all the applications currently downloadable from the Camera Mouse website [42] accessible from Menu Controller. The popular text input programs Midas Touch Keyboard and Staggered Speech, which were not accessible with the original version of Menu Controller, were of particular concern. While working through these and also trying to make the UI Automation-enabled Menu Controller work on other windows that did not previously work with Menu Controller (such as Windows Explorer and Internet Explorer), it was observed that, unlike the windows that support the Menu-API, windows that use an alternative menu type do not behave in a generic way. Tweaks are needed to make the menus visible from Menu Controller, in some cases on a per-application basis.

6 Conclusions and future work

The user studies showed that the techniques introduced in this paper can improve GUIs so that people with severe motion impairments, who use assistive input devices, can interact with the GUIs successfully. The most important contribution of this work was the concept of retrofitting GUIs via sliding menubars. The process is automatic and adaptive, reducing the need of assistance of caregivers with the software setup. The strategies to simulate clicking-and-dragging and clicking-and-holding interactions, provide visual feedback, and reduce accidental selection commands were also successful with users with motor impairments.

The effort in creating Menu Controller has focused on the Windows environment. However, employing image processing techniques similar to the systems mentioned in the Related Works section [26, 27, 29] could enable the development of an accessibility tool that is platform independent. While the pointer-targeting techniques described in the Related Works section 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. Another effort to improve Menu Controller would be to handle other types of user interface widgets in addition to menus.

Ongoing work with Camera Canvas involves adding additional features to the program, such as the much requested fill and clip-art stamps, and continuing to look into simple games. Not only were games popular with the users for their entertainment value, they also provided valuable information about the user. The games in Camera Canvas were used to recommend user interface settings by automatically analyzing the movement abilities of a user during the game. This use of gamification was a good step toward a method to determine automatically how to adapt an interface so that it can provide a user experience that, in some sense, is optimal. Analyzing game performance every so often could provide metrics on how users' abilities change over time. To avoid being intrusive, Camera Canvas would ask and not require users to play these performance-measuring games. This software feature may be particularly beneficial for individuals who suffer from a degenerative disease that increasingly limits their motion abilities. Future work in this area could be informed by the experience in designing gaze-controlled games reported in the literature [64, 65].

In Camera Canvas, other alternatives to traditional UI elements were experimented with, e.g., Choice Boxes and Move Arrows as alternatives to sliders. Future work could be in the same line as Menu Controller—trying to take these alternative elements out of Camera Canvas and generalizing them into tools that can be used with existing software to make it more usable. For example, instead of interacting with a slider on a webpage, a user could launch a tool that presented a set of Move Arrows. The user of a mouse-replacement system could then rely on the more usable Move Arrows, which would send the same commands as if they were using the scrollbar of their application.

Finally, the interaction techniques discussed here could be incorporated into the mouse-replacement systems currently used by people with severe motion impairments, for example, Camera Mouse. This would help users to determine the most appropriate interface settings and empower them to adjust the settings themselves instead of relying on a caregiver.

Acknowledgments The authors would like to thank the participants of the user studies for their valuable feedback in helping improve the software. They would also like to thank the reviewers of our PETRA 2011 submission [6] for their helpful comments, and Robin Berghaus, Mikhail Breslav, Samuel Epstein, Nathan Fuller, James Gips, Fletcher Hietpas, Eric Missimer, Tessa Skinner, Ashwin Thangali, Diane Theriault, Gordon Towne, and Zheng Wu for their assistance during user studies and throughout the project. The human-subject research study has been approved by the Boston University IRB, and all participants (and their parents, if they are minors) consented to participate and release their photographs for publication. Funding from the National Science Foundation (HCC grants IIS-0910908, IIS-0713229, and IIS-0855065) and from the Boston University Undergraduate Research Opportunities Program is gratefully acknowledged.

References

1. Betke, M.: Intelligent interfaces to empower people with disabilities. In Nakashima H., Augusto J.C., Aghajan H. (eds.), Handbook of Ambient Intelligence and Smart Environments. Springer, New York (2009)
2. Mcmurrough, C., Ferdous, S., Papangelis, A., Boisselle, A., Makedon, F.: A survey of assistive computing devices for cerebral palsy patients. In: The 5th ACM International Conference on Pervasive Technologies Related to Assistive Environments (PETRA 2011), Heraklion, Crete, Greece, pp. F1:1–F1:8. ACM, June (2012)
3. Keates, S.: Motor impairments and universal access. In: Stephanidis Constantine (ed.), The Universal Access Handbook. CRC Press, Cleveland, pp. 5–1–5–12 (2009)
4. Kim, W.-B., Kwan, C., Fedyuk, I., Betke, M.: Camera canvas: Image editor for people with severe disabilities. Technical Report 2008-010, Computer Science Department, Boston University, May (2008)
5. Kwan, C., Betke, M.: Camera canvas: Image editing software for people with disabilities. In: Proceedings of the 6th International Conference on Universal Access in Human–Computer Interaction: Users Diversity—Volume Part II (UAHCI'11), Orlando, Florida, pp. 146–154. Springer, Berlin, July (2011)
6. Paquette, I., Kwan, C., Betke, M.: Menu Controller: Making existing software more accessible for people with motor impairments. In The 4th ACM International Conference on Pervasive Technologies Related to Assistive Environments (PETRA 2011), Heraklion, Crete, Greece, pp. 2:1–2:8. ACM, May (2011)
7. Worden, A., Walker, N., Bharat, K., Hudson, S.: 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, pp. 266–271 (1997)
8. Grossman, T., Balakrishnan, R.: 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, pp. 281–290 (2005)
9. Hurst, A., Mankoff, J., Dey, A.K., Hudson, S.E.: 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), pp. 183–186 (2007)
10. Spakov, O., Majaranta, P.: Scrollable keyboards for casual eye typing. PsychNol. J. **7**(2), 159–173 (2009)
11. Akram, W., Tiberii, L., Betke, M.: Designing and evaluating video-based interfaces for users with motion impairments. Universal Access in the Information Society. In review
12. Andrews, J.H., Hussain, F.: Johar: A framework for developing accessible applications. In: Proceedings of the 11th International ACM SIGACCESS Conference on Computers and Accessibility (Assets '09), pp. 243–244 (2009)

13. Olsen Jr., D.R., Hudson, S.E., Verratti, T., Heiner, J.M., Phelps, M.: 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), pp. 191–198 (1999)
14. Magee, J.J.: Adaptable Interfaces for People with Motion Disabilities. PhD thesis, Computer Science Department, Boston University, September (2011)
15. Wobbrock, J.O., Kane, S.K., Gajos, K.Z., Harada, S., Froehlich, J.: Ability-based design: concept, principles and examples. ACM Trans. Access. Comput. **3**, 9:1–9:27 (2011)
16. Shein, F.: Human interface design and the handicapped user. In: Proceedings of the Computer–Human Interaction Conference, pp. 292–293. ACM (1986)
17. Magee, J.J., Betke, M.: HAIL: hierarchical adaptive interface layout. In: K. Miesenberger et al., (eds.) 12th International Conference on Computers Helping People with Special Needs (ICCHP 2010), Vienna University of Technology, Austria, Part 1, LNCS 6179, pp. 139–146. Springer, Berlin, July (2010)
18. Magee, J.J., Epstein, S., Missimer, E., Betke, M.: Adaptive mappings for mouse-replacement interfaces. In: The 12th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS 2010), p. 3. Orlando, Florida, USA, October (2010)
19. Magee, J.J., Epstein, S., Missimer, E.S., Kwan, C., Betke, M.: Adaptive mouse-replacement interface control functions for users with disabilities. In Proceedings of the 6th International Conference on Universal Access in Human–Computer Interaction: Users Diversity—Volume Part II (UAHCI'11), Orlando, Florida, pp. 332–341. Springer, Berlin, July (2011)
20. Gajos, K.Z., Weld, D.S., Wobbrock, J.O.: Automatically generating personalized user interfaces with Supple. Artif. Intell. **174**, 910–950 (2010)
21. Connor, C., Yu, E., Magee, J., Cansizoglu, E., Epstein, S., Betke, M.: Movement and recovery analysis of a mouse-replacement interface for users with severe disabilities. In: Proceedings of the 13th International Conference on Human–Computer Interaction (HCI International 2009), pp. 1–10. San Diego, CA, July (2009)
22. Donegan M. (2012) Features of gaze control systems. In: Majaranta P., Aoki H., Donegan M., Hansen D.W., Hansen J.P., Hyrskykari A., Räihä K. (eds) Gaze Interaction and Applications of Eye Tracking: Advances in Assistive Technologies. IGI Global, pp. 28–34
23. Hutchings, D.R., Stasko, J.: Mudibo: multiple dialog boxes for multiple monitors. In: CHI '05 Extended Abstracts on Human Factors in Computing Systems, pp. 1471–1474 (2005)
24. Tan, D.S., Meyers, B., Czerwinski, M.: WinCuts: manipulating arbitrary window regions for more effective use of screen space. In CHI '04 Extended Abstracts on Human Factors in Computing Systems, pp. 1525–1528 (2004)
25. Stuerzlinger, W., Chapuis, O., Phillips, D., Roussel, N.: User Interface Façades: Towards fully adaptable user interfaces. In: Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST '06), pp. 309–318 (2006)
26. Amant, R., Riedl, M.O., Ritter, F.E., Reifers, A.: Image processing in cognitive models with SegMan. In: Proceedings of the 11th International Conference on Human–Computer Interaction (HCII International 2005), pp. 1–10. July (2005)
27. Hurst, A., Hudson, S.E., Mankoff, J.: Automatically identifying targets users interact with during real world tasks. In: Proceedings of the 14th International Conference on Intelligent User Interfaces (IUI'10), pp. 11–20 (2010)
28. Active Accessibility. Retrieved July 23, 2012, from http://msdn.microsoft.com/en-us/library/aa291313%28VS.71%29.aspx
29. Dixon, M., Fogarty, J.: 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), pp. 1525–1534 (2010)
30. Hornof, A.J., Cavender, A.: EyeDraw: Enabling children with severe motor impairments to draw with their eyes. In: Proceedings of ACM Conference on Human Factors in Computing Systems (CHI), pp. 161–170 (2005)
31. Harada, S., Wobbrock, J.O., Landay, J.A.: Voicedraw: a hands-free voice-driven drawing application for people with motor impairments. In: Proceedings of the 9th International ACM SIGACCESS Conference on Computers and Accessibility, Tempe, Arizona, pp. 27–34 (2007)
32. Betke, M., Gips, J., Fleming, P.: The Camera Mouse: Visual tracking of body features to provide computer access for people with severe disabilities. IEEE Trans. Neural Syst. Rehabil. Eng. **10**(1), 1–10 (2002)
33. Gorodnichy, D., Dubrofsky, E., Ali, M.: 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.
34. Manresa-Yee, C., Varona, J., Perales, F.J., Negre, F., Muntaner, J.J.: Experiences using a hands-free interface. In: Proceedings of the 10th International ACM SIGACCESS Conference on Computers and Accessibility, pp. 261–262, New York, NY, (2008). ACM
35. SmartNAV by NaturalPoint. Retrieved July 23, (2012), http://www.naturalpoint.com/smartnav
36. The COGAIN website. Maintained by the Communication by Gaze Interaction Association for promotion of research and development in the field of gaze-based interaction in computer-aided communication and control. Retrieved July 23, 2012, http://www.cogain.org
37. Ashmore, M., Duchowski, A.T., Shoemaker, G.: Efficient eye pointing with a fisheye lens. In: Proceedings of Graphics Interface (GI '05), pp. 203–210 (2005)
38. The COGAIN eye-tracker website. List of commercial eye tracking systems used for controlling a computer or as communication aids by people with disabilities and open-source and freeware software for gaze and eye tracking and eye movement analysis. Retrieved July 23, (2012), http://www.cogain.org/wiki/Eye_Trackers
39. Majaranta, P.:Communication and text entry by gaze. In: Majaranta P., Aoki H., Donegan M., Hansen D.W., Hansen J.P., Hyrskykari A., Räihä K. (eds) Gaze Interaction and Applications of Eye Tracking: Advances in Assistive Technologies, pp. 63–77. IGI Global (2012)
40. Majaranta, P., Bates, R., Donegan, M.: Eye tracking. In: Stephanidis Constantine (eds.), The Universal Access Handbook. CRC Press, Cleveland, pp. 36–1-36–17 (2009)
41. Prendinger, H., Hyrskykari, A., Nakayama, M., Istance, H., Bee, N., Takahasi, Y.: Attentive interfaces for users with disabilities: eye gaze for intention and uncertainty estimation. Univers. Access Inf. Soc. **8**(4), 339–354 (2009)
42. The Camera Mouse website. Retrieved July 23, 2012, http://www.cameramouse.org (2012)
43. Jacob, R.J.K.: What you look at is what you get. Computer **26**(7), 65–66 (1993)
44. Dragon speech recognition software. Retrieved July 23, 2012, from http://www.nuance.com/dragon
45. Windows Media Player. Windows Media Player—Microsoft Windows. Retrieved July 23, 2012, from http://windows.microsoft.com/en-US/windows/products/windows-media-player
46. Windows development (Windows). Retrieved July 23, 2012, from http://msdn.microsoft.com/en-us/library/ee663300%28v=VS.85%29.aspx

47. Calculator—Windows 7 features - Microsoft Windows. Retrieved July 23, 2012, from http://windows.microsoft.com/en-US/windows7/products/features/calculator
48. Microsoft Corporation. Menu functions. Retrieved July 23, 2012, from http://msdn.microsoft.com/en-us/library/ff468865%28v=VS.85%29.aspx, December (2010)
49. UI Automation Overview. Retrieved July 23, 2012, from http://msdn.microsoft.com/en-us/library/ms747327.aspx
50. Internet Explorer - Microsoft Windows. Retrieved July 23, 2012, from http://windows.microsoft.com/en-US/internet-explorer/products/ie/home
51. Mozilla Firefox web browser. Retrieved July 23, 2012, from http://www.mozilla.com/en-US/firefox/fx/
52. The Facial Analysis Homepage. Retrieved July 23, 2012, http://mambo.ucsc.edu/psl/fanl.html (2012)
53. Ponweiser, W., Vincze, M.: Task and context aware performance evaluation of computer vision algorithms. In: International Conference on Computer Vision Systems: Vision Systems in the Real World: Adaptation, Learning, Evaluation. Bielefeld, Germany (ICVS 2007) (2007)
54. Varona, J., Manresa-Yee, C., Perales, F.J.: Hands-free vision-based interface for computer accessibility. J. Netw. Comput. Appl. **31**(4), 357–374 (2008)
55. Porta, M., Ravarelli, A., Spagnoli, G.: ceCursor, a contextual eye cursor for general pointing in windows environments. In: Proceedings of the 2010 Symposium on Eye-Tracking Research and Applications (ETRA '10), pp. 331–337 (2010)
56. Kim, H., Ryu, D.: Computer control by tracking head movements for the disabled. In: 10th International Conference on Computers Helping People with Special Needs (ICCHP), Linz, Austria, LNCS 4061, pp. 709–715. Springer, Berlin (2006)
57. Kjeldsen, R.: Improvements in vision-based pointer control. In: Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility, pp. 189–196, New York, NY (2006). ACM
58. Loewenich, F., Maire, F.: Hands-free mouse-pointer manipulation using motion-tracking and speech recognition. In: Proceedings of the 19th Australasian Conference on Computer-Human Interaction (OZCHI), pp. 295–302, New York, NY, (2007). ACM
59. Palleja, T., Rubion, E., Teixido, M., Tresanchez, M., del Viso, A.F., Rebate, C., Palacin, J.: Simple and robust implementation of a relative virtual mouse controlled by head movements. In Proceedings of the Conference on Human System Interactions, pp. 221–224, Piscataway, NJ, 2008. IEEE.
60. Tu, J., Tao, H., Huang, T.: Face as mouse through visual face tracking. Computer Vision and Image Understanding **108**(1-2), 35–40 (2007)
61. Dawe, M.: 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, pp. 1143–1152. New York, NY, USA, (2006). ACM
62. Camera Mouse technology reaches 100,000th download milestone. Retrieved July 23, 2012. http://www.bc.edu/publications/chronicle/TopstoriesNewFeatures/features/cameramouse030410.html, published March 2010
63. The Camera Mouse Suite website. The Camera Mouse Suite is the "beta version" or "research version" of Camera Mouse. It is free and provides a suite of application programs. http://cameramouse.bu.edu (2012)
64. Isokoski, P., Joos, M., Spakov, O., Martin, B.: Gaze controlled games. Univers. Access Inf. Soc. **8**(4), 323–337 (2009)
65. Istance, H., Hyrskykari, A., Immonen, L., Mansikkamaa, S., Vickers, S.: Designing gaze gestures for gaming: An investigation of performance. In Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications, ETRA '10, pp. 323–330. ACM (2010)