Tools for a Gaze-Controlled Drawing Application – Comparing Gaze Gestures against Dwell Buttons

Henna Heikkilä

Tools for a Gaze-controlled Drawing Application – Comparing Gaze Gestures against Dwell Buttons

Henna Heikkilä

Tampere Unit for Computer-Human Interaction
School of Information Sciences
University of Tampere, Finland

henna.heikkila@sis.uta.fi

Abstract. We designed and implemented a gaze-controlled drawing application that utilizes modifiable and movable shapes. Moving and resizing tools were implemented with gaze gestures. Our gaze gestures are simple one-segment gestures that end outside the screen. Also, we use the closure of the eyes to stop actions in the drawing application. We carried out an experiment to compare gaze gestures with a dwell-based implementation of the tools. Results showed that, in terms of performance, gaze gestures were an equally good input method as dwell buttons. Furthermore, more than 40% of the participants gave better ratings for gaze gestures than for the dwell-based implementation, and under 20% preferred dwell over gestures. Our study shows that gaze gestures can be a feasible alternative for dwell-based interaction when they are designed properly and implemented in the appropriate application area.
Keywords: gaze interaction, eye tracking, drawing with gaze, gaze gestures

1 Introduction

Eye trackers and gaze-controlled applications enable many disabled users to join the information society independently. Gaze-controlled applications are controlled via eye gaze through an eye tracker. Before use, the eye tracker is calibrated to the user's eyes. Then, during use, the eye tracker follows the user's gaze point and delivers the data to applications. The applications use the data to determine the user's point of interest or intentions. Often, the applications have dwell buttons that the user clicks to control the application. A dwell button is "clicked" when the user's gaze point has remained on the button for a predetermined time (usually 200–500 milliseconds).

Over 30 years, eye tracking research has concentrated mostly on communication. In the last decade, the focus has shifted towards leisure applications, such as games and online communities. Many researchers work to enable disabled users to use applications similar to those that able-bodied users already use. Among these are writing applications, Internet browsers, drawing applications, and games of various types. We have concentrated on drawing applications, and our goal is to implement a

drawing application that is easy to use and enables the user to correct their drawing mistakes easily.

Even when fixating on a target, the eye does not stay still. The small natural movements occurring during fixation are called microsaccades. Microsaccades usually stay within one degree of movement, which translates to remaining within one square centimeter on the screen when the user is sitting at arm's length from the monitor. The jitter caused by the microsaccades makes it difficult to hit small objects. Also, small calibration errors are common, especially after one has been using the tracker for a while. To avoid the problems caused by microsaccades and calibration errors, the controls, such as buttons and menus, need to be larger in gaze-controlled applications than normally. Larger controls take room from the screen, particularly in drawing applications, where most of the screen space is needed for the drawing area.

In gaze-controlled applications, the user uses the eyes to study the feedback the application is giving and to control the application. If these two actions are not well separated from each other, the "Midas Touch problem" may arise: wherever the user looks, a command is issued [11]. This problem is often rectified by extending the dwell time used for determining whether a command has been issued or not.

At first, buttons clicked through dwell time and blinking were the most commonly used input method for issuing commands. A little over 10 years ago, the first gesture-like input methods for gaze-controlled applications were introduced, see [3]. The concept of gaze gestures was first put forward five years ago [1], and it has evolved since. Fundamentally, gaze gestures are predetermined gaze paths, or patterns of eye movements, that are interpreted as commands to the application. Gaze gestures can be short or long, simple or more complex, location-bound or location-independent.

Next, we give an overview of research on gaze-controlled drawing applications and on gaze gestures that are relevant to our research. We then present our gaze-controlled drawing application, called EyeSketch, and show how gaze gestures are used in its tools. Towards the end of the piece, we present the study wherein we compared dwell time to gaze gestures, and we discuss the results. We conclude by considering the work done so far and discussing some future directions for our drawing application.

2 Related Research

To our knowledge, five drawing applications controlled via eye gaze have been presented previously. Four of them utilize only gaze, and the other combines gaze with voice commands.

The first eye drawing application, Eye Painting (also known as EaglePaint), was presented 16 years ago by Gips and Olivieri [2]. Eye Painting was one of the applications for their EagleEyes, an EOG-based eye tracking technology. In Eye Painting, the user was able to draw colored lines on the screen by moving the head and eyes.

Eye Painting utilizes so-called free-eye drawing [16], in which the line of drawing appears wherever the user looks and the person drawing has no way of lifting the pen

from the drawing canvas. Thus, every shape is connected to the next by the line. A related problem with free-eye drawing is the lack of separation between drawing and looking around. When gaze is used to control a technology, usually the same channel is used for input and for examining the output. If users want to look around and examine the drawing, they probably want to pause the drawing process for the time being.

To solve the aforementioned problems with free-eye drawing, Hornof et al. [6, 7, 8] created an application called EyeDraw. In EyeDraw, looking and drawing are separated by two 500-millisecond dwell-time spans; drawing of a shape starts only when the gaze point has stayed relatively still for a second [7]. To end the drawing of the shape, the user needs to dwell for a second at the end point. Instead of free-eye drawing, EyeDraw utilizes shapes. In the first version, the user was able to draw only lines and ellipses, but the shape collection later grew to include rectangles and predefined stamps too. EyeArt [14] resembles EyeDraw in many respects, but it has a wider range of drawing tools, including seven different shapes, a text tool, and an eraser tool. In addition, the user can adjust the border thickness of a soon-to-be-drawn shape and fill drawn shapes with color by using the paint can tool.

Van der Kamp and Sundstedt [17] presented a drawing application that combines gaze and voice input. A voice command is used in place of dwelling to control drawing and to access the tools and their properties. These authors claim that their solution solves two problems that plagued the previous applications. First, they wanted to remove the need to dwell, since, they said, the use of dwell frustrates users. Second, they hid the tool menus to free screen space for the drawing canvas and to prevent unintended selections from the menus during drawing or looking. In their solution, tool menus appear only through a voice command.

The three drawing applications discussed above utilize separation of looking and drawing. They share the problem that the position or size of the shape drawn cannot be adjusted, which means that the user needs to draw the shape in precisely the right place and in exactly the right size or else undo/erase it and start again. Yeo and Chiu [18] introduced a third technique when designing their gaze-estimation model that constitutes an attempt to separate looking (or thinking and searching) from drawing through examination of gaze patterns. Their model assumes that when the gaze points are close together, the user wants to draw, and that when the gaze points are mostly far from each other, the user is thinking or searching for something. When the gaze points cluster within an area, that area is determined to be the "area of interest." If a fixation longer than 500 milliseconds falls within the area of interest, the centroid of the gaze points is calculated and drawing begins at that point.

In our drawing application, gaze gestures are used to move and resize shapes. Our gaze gestures utilize simple, one-segment gestures and off-screen space. Next, we present the three gaze-gesture implementations that are relevant for our study. Møllenbach et al. [15] created simple, single-segment gaze gestures. In their one-segment gaze gestures, the gesture was made across the screen: it started from what they called the gesture area and ended in another gesture area, on the opposite side of the screen. The assortment of these Single Gaze Gestures, as the authors call

them, is small, but they can be used for simple tasks, such as top-level navigation of applications or controlling one's environment.

Isokoski [9] used off-screen targets in his eye writing application. He used five off-screen targets attached to the monitor frame. The user's gaze was tracked with a head-mounted SMI EyeLink tracker, which was able to track the gaze beyond the screen area when the user was seated 100 centimeters away from the monitor. A short dwell time, 100 milliseconds, was used as the threshold for determining whether the user's gaze actually stopped over an off-screen target or just wandered over it.

Another application using off-screen space is Snap Clutch, by Istance et al. [10]. In Snap Clutch, the user can switch mode in the application by looking outside the screen space. The authors claim that quick glances of this sort are a fast and effortless way to control their application.

In our application, we use the closure of the eyes also. The closure of both eyes for a longer time is a rarely used input method, although it is easier to recognize as intentional than are the more frequently used blinks and winks (closure of one or both eyes for only a short time). Especially in the case of blinks, it is difficult to determine which are intentional and which involuntary – reflexive actions that occur when the eyes are getting dry, as the eyes often do more readily when one is looking at a computer screen. As far as we know, only Hemmert et al. [4, 5] have used the closure of one and both eyes to control applications. By closing both eyes, the user was able to activate text-to-speech functionality in a writing application, and closing just one eye allowed users to switch between modes in a first-person shooter game and to filter other than the most recently used icons from the desktop.

3 EyeSketch: A Gaze-controlled Drawing Application

Our motivation in the design of our drawing application is twofold. First, we wanted to create a drawing application with which the user can produce pleasing pictures without unintentional gaps between shapes or accidental overlapping of shapes. Second, we wanted to create a new kind of drawing application. Of the preexisting drawing applications, none used objects that can be modified, yet able-bodied users have their choice of many such applications. Moreover, we believe that modifiable objects can solve the positioning problem in addition.

3.1 The application

We chose an approach wherein the shapes, or objects, drawn can be moved or resized, and in which their other properties can be modified after these are drawn.

For the first version of our application, we implemented basic drawing tools, tools for modifying the shapes drawn, and tools for saving and opening pictures drawn with the application. Tool buttons were placed around the drawing canvas and implemented as dwell buttons. We used 80 × 80 pixels as the size for a tool button. These buttons are selected when the gaze point has stayed on the button for 400 milliseconds.

Our basic drawing tools include tools for drawing rectangles, ellipses, and lines. Before and after drawing of the shapes, their fill color, border color, and border thickness can be changed. To aid in creation of the drawing, a grid is implemented behind the drawing canvas. The user can choose whether to display the grid or not.

For later modification of a shape, we have a *Select* tool. When a shape is selected, its color and line thickness can be changed, and it can be removed with the *Undo* tool.

The *Move*, *Nudge*, and *Resize* tools are implemented with gaze gestures. With the *Nudge* tool, the shape moves one grid square in the direction of the gaze gesture made. With the *Move* tool, the shape starts to move towards the gesture's direction and stops when the user closes the eyes or when the moving shape hits the edge of the drawing canvas. The *Resize* tool causes resizing handles to appear at the sides of the selected shape (see Fig. 1). The size of each handle is 50 × 50 pixels. A handle is selected by dwell first (100 milliseconds), followed by a gesture; depending on the gesture direction (inwards or outwards), the shape shrinks or grows from the side on which the handle is attached.

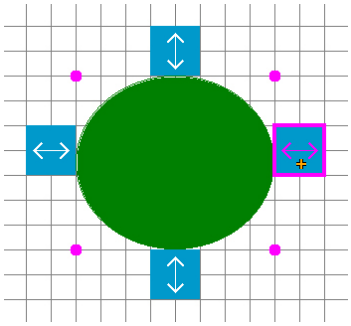

Fig. 1. The resizing handles for the *Resize* tool appear around the selected shape.

We integrated the COGAIN ETU Driver[1] into the drawing application to deliver the eye tracking data from the eye tracker to our drawing application. The ETU Driver supports several makes of eye tracker. Therefore, the application can be used with multiple eye trackers, since the ETU Driver makes sure that the gaze data will be delivered to the application in the same form regardless of the eye tracker used.

3.2 Gaze gestures

We chose gaze gestures for this use since they are less vulnerable to calibration errors and the jitter in the eyes. Our gaze gestures are simple one-segment gestures that end outside the screen. We also use the closure of both eyes to stop a moving shape.

The gaze gesture used with the *Move* and *Nudge* tools always starts on top of a shape already drawn, proceeds in one of the eight directions (toward a side or corner of the screen), and ends outside the screen (see Fig. 2). We named the eight directions after the cardinal and ordinal directions, with north being toward the top of the screen, northeast toward the upper right-hand corner of the screen, east toward the right side of the screen, etc.

With the *Resize* tool, the gaze gesture starts on top of a resize handle attached to the side of the selected shape; proceeds left/right or up/down, depending on the side on which the handle is attached; and ends outside the screen area.

[1] The COGAIN ETU Driver (i.e., Eye-Tracking Universal Driver) can be downloaded from *http://www.sis.uta.fi/~csolsp/downloads.php*.

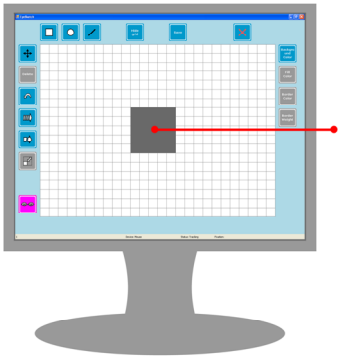

Fig. 2. The gaze gesture starts on top of a drawn shape (or a resizing handle) and ends outside the screen.

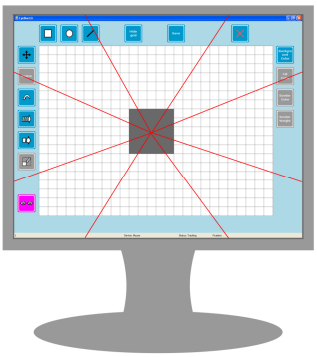

Fig. 3. At least three gaze points must fall into the same gesture segment before the gesture can be completed.

To be able to start the gaze gesture, the gaze has to stay on the shape or resize handle for 100 milliseconds for it to become selected. The user can start the gesture when the gaze cursor changes its color from black to orange. As the user makes the gaze gesture, three gaze points must fall into the same segment (see Fig. 3) in the direction of the gesture before exiting the screen area. Since the 60 Hz eye trackers take a gaze-point sample once every 16th millisecond, the move from the shape to outside the screen must take at least 64 milliseconds. If it takes more than 1,500 milliseconds, the gesture process stops. When the gaze has remained outside the screen area for 100 milliseconds, the gesture-recognition process ends and the command is issued. A feedback sound is played to the user when they can return the gaze to the screen without canceling the action.

In use of the *Move* tool, the gaze gesture makes the shape move in the direction of the gesture. Closing both eyes for 300 milliseconds stops the movement. While the eyes are closed, the shape keeps moving until the threshold time has been reached. Once that time has elapsed, the shape returns to where it was when the eyes were closed. A feedback sound is played to the user when the eyes may be opened.

4 Gaze Gestures vs. Dwell – an Experiment

To find out whether our gaze gestures would be a feasible input method for the *Move*, *Nudge*, and *Resize* tools, we designed an experiment in which the gaze gestures and often-used dwell buttons were compared.

4.1 Participants

Twelve participants, seven male and five female, volunteered for the tests. Their ages ranged from 18 to 38 years (mean: 24.3 years). Only one of the participants wore eyeglasses during the test. None of the participants had prior experience of eye tracking. Nine participants were familiar with the concept of gestures, and five of

them had tried gestures in some form; they reported having used them with cell phones (hand/finger gestures) and with video-game consoles (bodily gestures).

4.2 Apparatus

We used a Tobii T60 eye tracker with a sampling rate of 60 Hz to track the participant's gaze. The resolution of the screen was set to 1280 × 1024 pixels (17-inch LCD screen with a width of 338 mm and height of 272 mm).

The test application was a light version of our drawing application: only the *Move*, *Nudge*, *Resize*, *Undo*, and *Look around* tools were available. In each task, the object to be moved or resized was already drawn in the drawing area. The target size and position were indicated through a similar object with a thick red border (see Fig. 4).

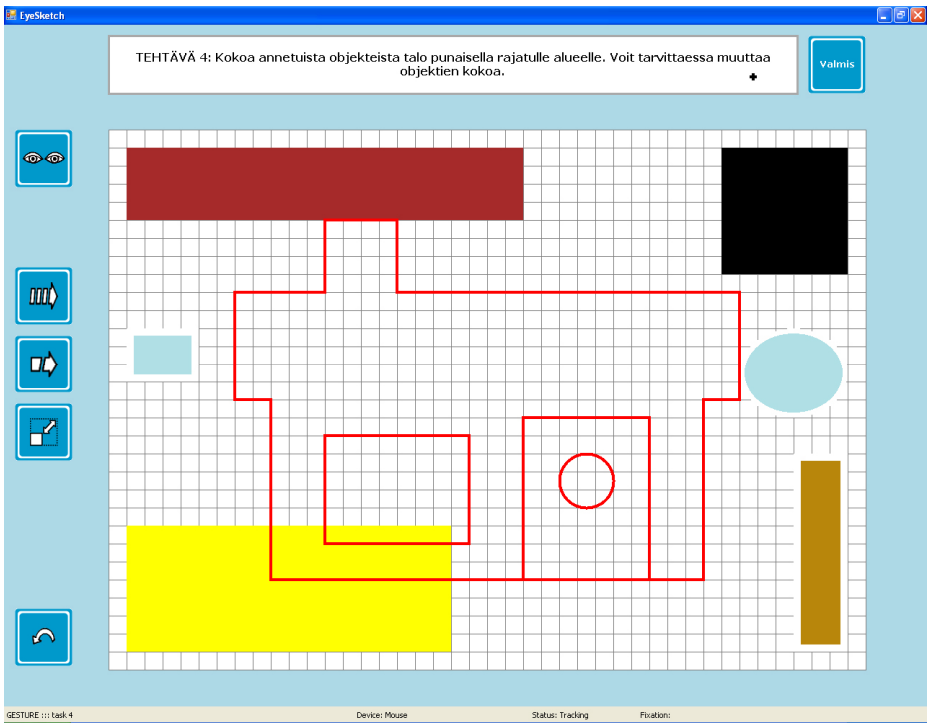

Fig. 4. Test application showing Task 4: *Use drawn objects to build a house in the target area. Resize the objects when necessary*.

4.3 Dwell Implementation

The dwell implementation differs from our gaze-gesture implementation described above only in terms of the implementation of the *Move*, *Nudge*, and *Resize* tools.

With the *Move* and *Nudge* tools, eight dwell buttons, with arrows showing the direction appear, around the selected shape (as shown in Fig. 5). The participant needs to keep the gaze on the button for 400 milliseconds for it to be clicked. The *Move* tool makes the shape start to move in the given direction when the arrow button is clicked. The movement is stopped in the same way as in the gesture implementation: by closing of both eyes. In the implementation of the *Nudge* tool, the shape moves one grid step in the given direction and stops automatically.

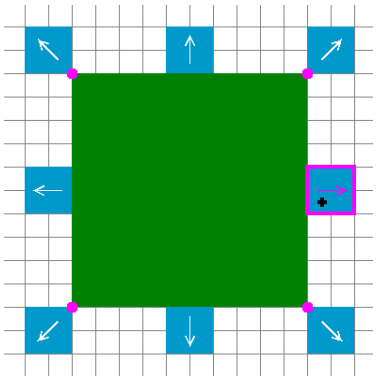

Fig. 5. The dwell buttons for the *Move* tool appear around the selected shape in the dwell implementation.

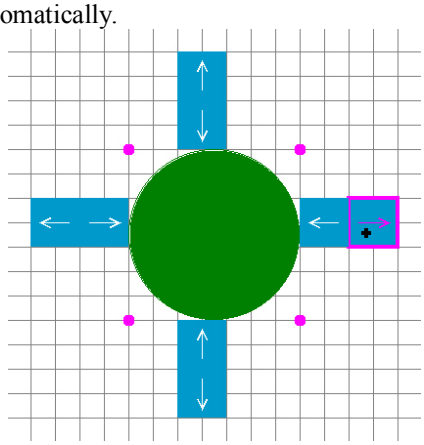

Fig. 6. The dwell buttons for the *Resize* tool appear around the selected shape in the dwell implementation.

For the *Resize* tool, a pair of dwell buttons appears on each side of the selected shape (see Fig. 6). The button closer to the shape makes the shape smaller, and the one further from the shape increases the shape's size. When the participant has fixated on the dwell button for 400 milliseconds, it will be clicked and the size of the shape decreases or increases by one step.

The 400-millisecond threshold for dwell time was selected on the basis of literature from the field of eye typing, wherein dwell time is used to select letters from an on-screen keyboard. Majaranta and Räihä [12] concluded in their review that in eye typing studies with novice users, dwell times ranged from 450 to 1000 milliseconds. Majaranta et al. [13] performed a longitudinal eye typing study wherein participants were able to adjust the dwell time. In their study, none of the participants used a dwell time of 400 milliseconds or less during the first session. After five sessions (that is, 75 minutes' practice), most participants had decreased the dwell time to 400 milliseconds or less. For novice users, any dwell time shorter than 400 milliseconds would cause significantly more unintended commands.

4.4 Tasks

We asked participants to perform four tasks with each style of input. The tasks were the following:

1. Move the object drawn to the target area.
2. Use already-drawn objects to build a house in the target area.
3. Resize the object drawn until it matches the target area.
4. Use drawn objects to build a house in the target area. Resize the objects when necessary.

For Tasks 1 and 2, the participant had only movement tools, *Move* and *Nudge*, available. For Task 3, only the *Resize* tool was active. For the fourth task, the participant was able to use both the movement tools and the resizing tool. Tasks 1 and 3 were used to train the participants in use of the new tools. Task 2 was selected to reveal possible difficulties when there are several objects in the drawing area. With Task 4, we wanted to see how well switching from one tool to another works.

4.5 Procedure

Each test took 40–60 minutes. At the beginning of the test, the participant was asked to fill in a questionnaire form, for background information. Then the purpose and the procedure of the test were introduced, and informed consent was requested from the participant. The test had two parts, each using one of the two input styles. The two parts followed the same procedure; only the input style was different. The order of the input styles was counterbalanced.

At the beginning of each part, the experimenter demonstrated the input style with a mouse. Then the participant was seated in front of the eye tracker, at arm's length from the monitor, and the eye tracker was calibrated. After calibration, the experimenter started the testing software and the participant performed the four tasks. After completing the tasks, the participant was asked to fill in a user-satisfaction form. Meanwhile, the experimenter restarted the eye tracker. After a short break, the second part of the test was started. After the second part and the associated user-satisfaction form, the participant was briefly interviewed about the experiences during the test.

4.6 Results and Discussion

We calculated task-completion times, times for completing an action, and the number of unnecessary actions from the data collected. To test the statistical significance of our results, we used repeated-measures ANOVAs with Greenhouse–Geisser correction and paired-sample *t*-tests for *post hoc* pairwise comparisons.

Task-completion times. In comparison of the mean times for completion of a full task (see Fig. 7), the dwell implementation was revealed to be faster for tasks 1 and 2, wherein the participants only had to move objects. In tasks 3 and 4, which involved the need to resize the objects in addition, the two implementation types took equally long for completion, on average. This means that the resizing actions take so much longer to complete with the dwell implementation that the advantage gained in the moving actions is lost. Only the main effect for the task was significant ($F_{2,19} = 72.45$, $p < .001$), as can be expected from the nature of the tasks.

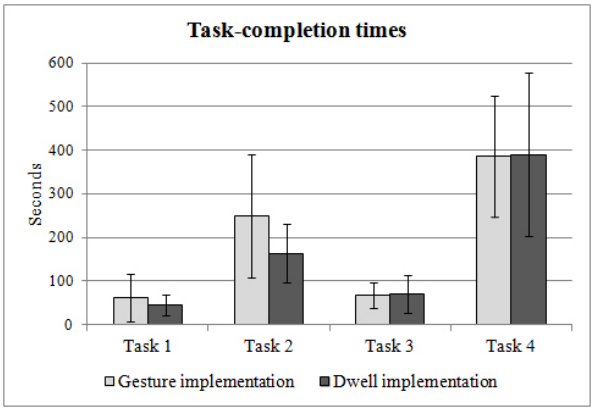

Fig. 7. Task-completion times, in seconds, for the two implementations. The error bars show the standard deviations of the means.

When examining the times to complete the first task with the two implementation types, we found that the first performance of Task 1 took significantly more time than the second one ($F_{1,10} = 5.95$, $p < .05$). This was independent of the implementation type used ($F_{1,10} = 1.47$, $p > .05$). The results demonstrate that it always takes time to figure out how to use one's eyes to control an application when a gaze-controlled application is used for the first time.

Completion time for an action. On average, performance of an action was almost equally fast in the two implementation types (see Fig. 8) for all actions. Statistical tests showed that task had a significant effect on the time taken per action ($F_{1,15} = 8.97$, $p < .01$). It also had an interaction effect with the implementation type on the completion times ($F_{1,15} = 7.96$, $p < .01$). Implementation type on its own did not have a statistically significant effect on completion times ($F_{1,11} = 1.19$, $p > .05$).

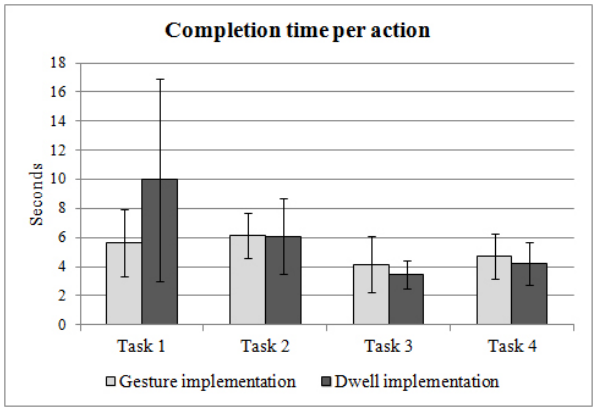

Fig. 8. Completion time per action, in seconds, for the two implementations. The error bars show the standard deviations of the means.

The exception is that in Task 1, the actions with the dwell implementation took almost twice as long as those performed with the gesture implementation. The implementation type had a significant effect on completion time for Task 1 ($F_{1,10} = 5.67$, $p < .05$), and it did not matter which implementation type the participant used first ($F_{1,10} = 0.00$, ns). We believe this result reflects the fact that the participants tried out the gaze gestures more than the dwell buttons before starting to perform the task. We observed in the tests that many participants made several gestures to get a shape to move and then stopped the movement before the shape had moved more than a couple of steps. With the dwell implementation, there was less behavior of this kind.

Excess actions. We calculated the optimal number of actions for each task. Optimal performance in tasks 1–4 involved 2, 12, 10, and 54 actions, respectively. Only four times during the tests did a participant manage to complete a task optimally. When the task was only to move the shapes (tasks 1 and 2), the participants used more actions in the gesture implementation than in the dwell implementation. However, when the tasks included resizing of the shapes (tasks 3 and 4), more actions were employed in the dwell implementation than in the gesture implementation.

The data support our observations during the tests: the participants had difficulties in resizing the shapes with the dwell implementation, because the dwell buttons to make the shape smaller and larger were next to each other. Because of jitter in the gaze and small calibration errors, the participants often accidentally clicked the wrong dwell button. Then another action was needed to reverse this wrong action. Therefore, to complete one successful action, the participant had to perform three actions.

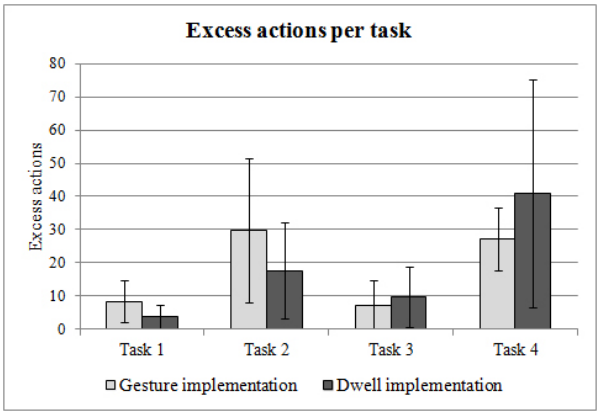

Fig. 9. Number of excess actions per task in the two implementations. The error bars show the standard deviations of the means.

As expected, task had a significant main effect on unnecessary actions ($F_{2,23} = 16.83$, $p < .001$), since the tasks were very different in how many actions were needed for their completion. Implementation type did not have a main effect on the number of excess actions ($F_{1,11} = 0.005$, ns), since whether a given implementation type fared better or worse varied from one task to the next. Statistical testing revealed that task

and implementation type had a significant interaction effect ($F_{2,20} = 4.73$, $p < .05$). This supports what is visible in Figure 9. The result means that one implementation type is better for certain tasks than the other, and vice versa.

The participants used more actions than needed in their very first task, no matter which implementation type they started the test with. When the participants started with the gesture implementation, they used, on average, 11.3 actions more than needed for completion of the first task. When facing the same task in the dwell implementation later, they used only 2.3 actions more than the optimum. The participants who started with the dwell implementation performed 5.0 actions more than the number needed in their very first task, and only 5.5 actions more when they later completed the task with the gesture implementation.

The statistical tests showed that there was a statistically significant difference between the two types of implementation in the number of unnecessary actions for Task 1 ($F_{1,10} = 6.10$, $p < .05$). That is, for Task 1, the participants made more unnecessary actions with one of the implementation types (the gesture implementation) than in the other. As described above with regard to completion time per action, we observed during the tests that, with the first task, the participants tried out the use of gaze gestures more than they did the use of dwell buttons. Our results suggest that the use of gaze gestures needs more training in the beginning than that of dwell buttons.

Subjective impressions. We asked the participants to evaluate their use experience on a seven-point Likert scale (with 1 indicating "strongly disagree" and 7 standing for "strongly agree"). They evaluated their experience on seven dimensions after using both implementation types. When one looks at the average scores, the gesture implementation appears better or at least equally good on all dimensions (see Fig. 10). The largest differences emerged for ease of resizing objects and in interaction speed. For ease of moving objects and on the natural-interaction dimension, the two implementation types were equally good. None of the differences was shown to be statistically significant in Wilcoxon signed-rank testing.

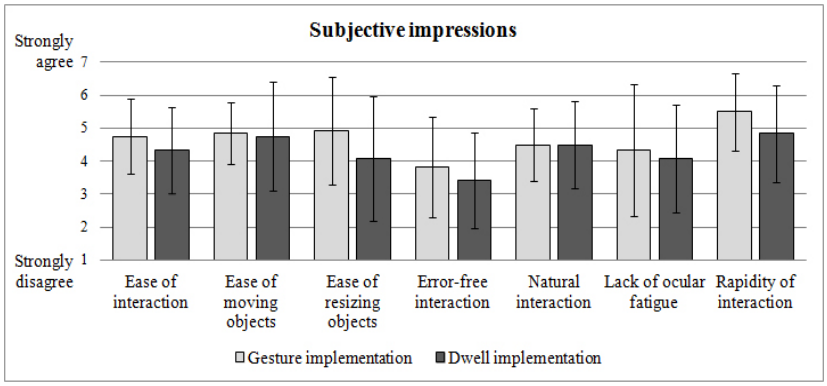

Fig. 10. Subjective impressions gathered after each implementation type. The error bars show the standard deviations of the means.

All participants who started with the dwell implementation rated the gesture implementation better than the dwell implementation or equally good. Participants who started with the gesture implementation were less unanimous with their scores. When we asked about the preference between the two in the interviews, seven participants preferred gaze gestures, four preferred the dwell implementation, and one was undecided. In particular, the difficulty in hitting the correct resizing dwell button in tasks 3 and 4 tipped the scale to gesture implementation. If we had increased the dwell-button size from 50 × 50 pixels or left space between the dwell buttons, such problems might have been avoided. However, seeing the drawing and accessing the objects drawn are essential to drawing applications. Also, in that solution, when several objects are placed close to each other and the user could readily select the wrong one by accident, the resize buttons for the wrongly selected object might hide the intended object and cause a dilemma. In our study, we already gave twice as much space from the drawing canvas to the dwell buttons as in the gesture implementation. Had we given them even more space, the situation would have been neither comparable to the gaze-gesture implementation nor appropriate for drawing applications anymore.

5 Conclusions

As described at the start of the paper, our motivation is to solve problems in existing gaze-controlled drawing applications by creating a new kind of drawing application that utilizes movable, resizable, and modifiable objects. The first step was to establish a functional way to move and resize the objects. For this purpose, we selected gaze gestures, since we wanted to keep the drawing canvas as free of buttons as possible.

The next step was to test whether the gaze gestures could work as well as the dwell buttons that are the traditional input style. The results from the experiment described in this paper are very encouraging. Although the dwell buttons were the better input style for moving shapes, in resizing tasks the gaze gestures proved to be an even better input style, solving all the problems from which the dwell implementation suffered. Furthermore, the participants were able to move and resize the shapes with gaze gestures even when the drawing canvas was half-filled with various shapes. We were excited to learn that most of the participants in our experiment felt that the gaze gestures worked well and that they preferred the gaze gestures to the dwell-button implementation.

In terms of time to issue a command, gaze gestures may never beat dwell buttons. The real advantage of gaze gestures is their ability to remain functional despite calibration errors and low accuracy of the eye tracker. Our resizing task showed how vulnerable the dwell-time input is to even small accuracy problems. Overall, our results showed that the simple, one-segment gaze gestures can be used for tasks other than switching between modes. The only limitation for one-segment gaze gestures is the small size of the gesture vocabulary. However, with adequate planning, the use cases could be numerous.

We have implemented a very usable way to move and resize shapes in a drawing application. Our next two steps are user tests with users from our target user group – i.e., with disabled users – and releasing the drawing application for the public.

Acknowledgements

This work was supported by the Finnish Doctoral Program in User-Centered Information Technology (UCIT).

References

1. Drewes, H. & Schmidt, A.: Interacting with the Computer Using Gaze Gestures. In: Baranauskas, M.C.C et al. (eds.): INTERACT 2007, Part II. LNCS, Vol. 4663, pp. 475–488. Springer, Heidelberg (2007).
2. Gips, J., & Olivieri, P.: EagleEyes: An Eye Control System for Persons with Disabilities. In: 11th International Conference on Technology and Persons with Disabilities (CSUN'96). Los Angeles, USA (1996).
3. Heikkilä, H., & Räihä, K.-J.: Speed and Accuracy of Gaze Gestures. Journal of Eye Movement Research, 3(2):1, 1–14 (2009).
4. Hemmert, F., Djokic, D., & Wettach, R.: Spoken Words: Activating Text-To-Speech through Eye Closure. In: CHI'08 Extended Abstracts on Human Factors in Computing Systems, pp. 2325–2330. ACM, New York (2008a).
5. Hemmert, F., Djokic, D., & Wettach, R.: Perspective Change: A System for Switching Between On-screen Views by Closing one Eye. In: Working Conference on Advanced Visual Interfaces, pp. 484–485. ACM, New York (2008b).
6. Hornof, A. J., & Cavender, A.: EyeDraw: Enabling Children with Severe Motor Impairments to Draw with Their Eyes. In: SIGCHI Conference on Human Factors in Computing Systems, pp. 161–170. ACM, New York (2005).
7. Hornof, A., Cavender, A., & Hoselton, R.: Eyedraw: A System for Drawing Pictures with Eye Movements. In: 6th International ACM SIGACCESS Conference on Computers and Accessibility, pp. 86–93. ACM, New York (2004a).
8. Hornof, A., Cavender, A., & Hoselton, R.: EyeDraw: A System for Drawing Pictures with the Eyes. In: CHI'04 Extended Abstracts on Human Factors in Computing Systems, pp. 1251–1254. ACM, New York (2004b).
9. Isokoski, P.: Text Input Methods for Eye Trackers Using Off-Screen Targets. In: 2000 Symposium on Eye Tracking Research & Applications, pp. 15–21. ACM, New York (2000).
10. Istance, H., Bates, R., Hyrskykari, A., & Vickers, S.: Snap Clutch, a Moded Approach to Solving the Midas Touch Problem. In: 2008 Symposium on Eye Tracking Research & Applications, pp. 221–228. ACM, New York (2008).
11. Jakob, R. J. K.: The Use of Eye Movements in Human-Computer Interaction Techniques: What You Look At is What You Get. ACM Transactions on Information Systems 9(2), 152–169 (1991).
12. Majaranta, P. & Räihä, K-J.: Text Entry by Gaze: Utilizing Eye-Tracking. In: MacKenzie, I.S. & Tanaka-Ishii, K. (eds.) Text Entry Systems: Mobility, Accessibility, Universality, pp. 175–187. Morgan Kaufmann, San Francisco (2007).

13. Majaranta, P., Ahola, U., & Špakov, O.: Fast Gaze Typing with an Adjustable Dwell Time. In: 27th International Conference on Human Factors in Computing Systems (CHI '09), pp. 357–360. ACM, New York (2009).
14. Meyer, A., & Dittmar, M.: Conception and Development of an Accessible Application for Producing Images by Gaze Interaction - EyeArt. (2009). Retrieved from *http://wiki.cogain. info /images/d/da/EyeArt_Documentation.pdf*.
15. Møllenbach, E., Lillholm, M., Gail, A., & Hansen, J.P.: Single Gaze Gestures. In: 2010 Symposium on Eye Tracking Research & Applications, pp. 177–180. ACM, New York (2010).
16. Tchalenko, J.: Free-Eye Drawing. Point (11), 36–41 (2001).
17. Van der Kamp, J., & Sundstedt, V.: Gaze and Voice Controlled Drawing. In: 1st Conference on Novel Gaze-Controlled Applications, p. 9. ACM, New York (2011).
18. Yeo, A. W., & Chiu, P. C.: Gaze Estimation Model for Eye Drawing. In: CHI'06 Extended Abstracts on Human Factors in Computing Systems, pp. 1559–1564. ACM, New York (2006).