The Use of Eye Movements in Human-Computer Interaction Techniques: What You Look At is What You Get

ROBERT J. K. JACOB
Naval Research Laboratory

In seeking hitherto-unused methods by which users and computers can communicate, we investigate the usefulness of eye movements as a fast and convenient auxiliary user-to-computer communication mode. The barrier to exploiting this medium has not been eye-tracking technology but the study of interaction techniques that incorporate eye movements into the user-computer dialogue in a natural and unobtrusive way. This paper discusses some of the human factors and technical considerations that arise in trying to use eye movements as an input medium, describes our approach and the first eye movement-based interaction techniques that we have devised and implemented in our laboratory, and reports our experiences and observations on them.

Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Tools and Techniques—*user interfaces*; H.1.2 [**Models and Principles**]: User/Machine Systems—*human factors*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*input devices and strategies, interaction styles, user interface management system*

General Terms: Design, Human Factors

Additional Key Words and Phrases: Eye movements, eye tracking, human-computer interaction, state transition diagram, UIMS, input

1. INTRODUCTION

Current user-computer dialogues tend to be one sided, with the bandwidth from the computer to the user far greater than that from user to computer. A fast and effortless mode of communication from a user to a computer would help redress this imbalance. We therefore investigate the possibility of introducing the movements of a user's eyes as an additional input medium. While the technology for measuring eye movements and reporting them in real time has been improving, what is needed is appropriate *interaction techniques* that incorporate eye movements into the user-computer dialogue in a convenient and natural way. This paper describes research at NRL on developing such interaction techniques. It discusses some of the human factors and technical considerations that arise in trying to use eye movements as an input medium, describes our approach and the first eye

This work was sponsored by the Office of Naval Research.
Author's address: Human-Computer Interaction Lab., Naval Research Laboratory, Washington, D.C. 20375.

movement-based interaction techniques that we have devised and implemented in our laboratory, and reports our experiences and observations on them.

2. BACKGROUND

Methods for Measuring Eye Movements

Available techniques for measuring eye movements vary considerably; while none is perfect, some are considerably more suitable for user-computer interaction than others. First, note that our goal is to measure *visual line of gaze*, that is, the absolute position in space at which the user's eyes are pointed, rather than, for example, the position of the eye in space or the relative motion of the eye within the head; not all eye tracking techniques do this [16]. Since both eyes generally point together, it is customary to track only one eye.

The simplest eye tracking technique is electronic recording, using electrodes placed on the skin around the eye to measure changes in the orientation of the potential difference that exists between the cornea and the retina. However, this method is more useful for measuring relative eye movements than absolute position. Perhaps the least user-friendly approach uses a contact lens that fits precisely over the corneal bulge and is held in place with a slight suction. This method is extremely accurate, but suitable only for laboratory studies. More practical methods use remote imaging of a visible feature located on the eye, such as the boundary between the sclera and iris, the outline of the pupil, or the corneal reflection of a light shone at the eye. All these require the head to be held absolutely stationary (a bite board is customarily used), to be sure that any measured movement represents movement of the eye, not the head. However, by simultaneously tracking two features of the eye that move differentially (because of the difference in radius of curvature of the cornea and sclera), it is possible to distinguish head movements (the two features move together) from eye movements (the two move with respect to one another), and the head need not be rigidly fixed. This is currently the most practical method for use in a conventional computer-and-user setting, since the eye tracker sits several feet from the user, nothing contacts him or her, and the head need not be clamped. In our laboratory, we use an Applied Science Laboratories (Waltham, Mass.) Model 3250R eye tracker [10, 16]. Figure 1 shows the components of this type of eye tracker. It simultaneously tracks the corneal reflection (from an infrared light shining on eye) and the outline of the pupil (illuminated by same light). Visual line of gaze is computed from the relationship between the two tracked points.

Previous Work

While current technology for measuring visual line of gaze is adequate, there has been little research on *using* this information in real-time. There is a considerable body of research using eye tracking, but it has concentrated on

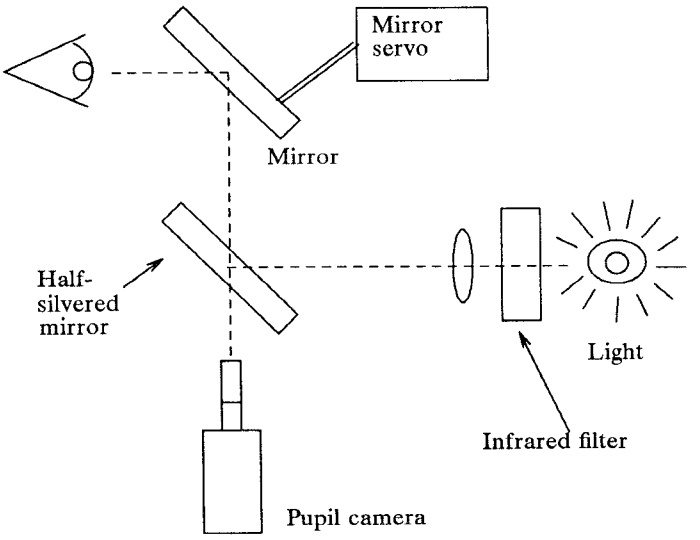

Fig. 1. Illustration of components of a corneal reflection-plus-pupil eye tracker The pupil camera and illuminator operate along the same optical axis, via a half-silvered mirror. The servo-controlled mirror is used to compensate for the user's head motions.

eye movement data as a tool for studying motor and cognitive processes [8, 11]. Such work involves recording the eye movements and then analyzing them subsequently; the user's eye movements do not have any effect on the computer interface while it is in operation.

Real-time eye input has been used most frequently for disabled (quadriplegic) users, who can use only their eyes for input (e.g., Hutchinson et al. and Levine [6, 9] report work for which the primary focus was disabled users). Our interest is, instead, on dialogues that combine real-time eye movement data with other, more conventional modes of user-computer communication. Bolt did some of the earliest work in this particular area and demonstrated several innovative uses of eye movements [1, 2, 14]. Glenn [5] used eye movements for several tracking tasks involving moving targets. Ware and Mikaelian [15] reported an experiment in which simple target selection and cursor positioning operations were performed approximately twice as fast with an eye tracker than with any of the more conventional cursor positioning devices. Fitt's law relationship as seen in experiments with other cursor positioning devices [4] remained true of the eye tracker; only the speed was different.

Characteristics of Eye Movements

To see an object clearly, it is necessary to move the eye so that the object appears on the fovea, a small area at the center of the retina. Because of this, a person's eye position provides a rather good indication (to within the

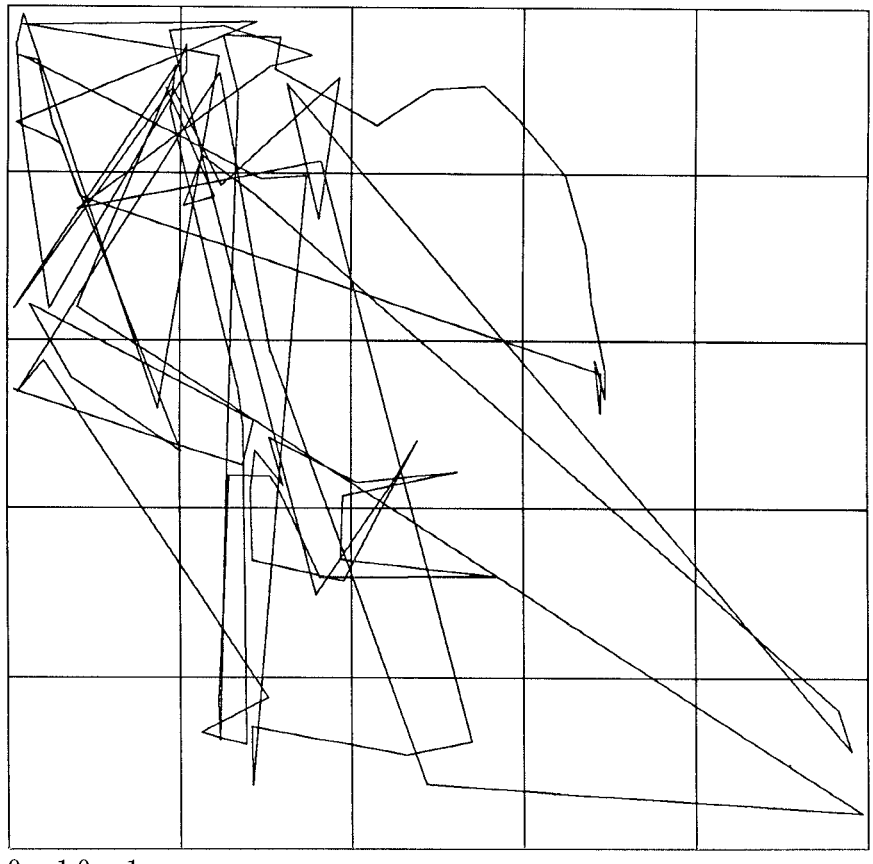

Fig. 2. A trace of a computer user's eye movements over approximately 30 seconds, while performing normal work (i.e., no eye-operate interfaces) using a windowed display. Jitter within each fixation has been removed from this plot.

one-degree width of the fovea) of what specific portion of the scene before him he is examining. The most common way of moving the eyes is a sudden, ballistic, and nearly instantaneous *saccade*. It is typically followed by a *fixation*, a 200-600 ms. period of relative stability during which an object can be viewed. During a fixation, however, the eye does not remain still; it still makes small, jittery motions, generally covering less than one degree. Smooth eye motions, less sudden than saccades, occur only in response to a moving object in the visual field. Other eye movements, such as nystagmus, vergence, and torsional rotation are relatively insignificant in a user-computer dialogue.

The overall picture of eye movements for a user sitting in front of a computer is a collection of steady (but slightly jittery) fixations connected by sudden, rapid saccades. The eyes are rarely entirely still. They move during a fixation, and they seldom remain in one fixation for long. Figure 2 shows a

trace of eye movements (with intra-fixation jitter removed) for a user using a computer for 30 seconds. Compared to the slow and deliberate way people operate a mouse or other manual input device, eye movements careen madly about the screen. During a fixation, a user generally thinks he is looking steadily at a single object—he is not consciously aware of the small, jittery motions. This suggests that the human-computer dialogue should be constructed so that it, too, ignores those motions, since, ultimately, it should correspond to what the user *thinks* he is doing, rather than what his eye muscles are actually doing. This will require filtering of the raw eye position data to eliminate the high-frequency jitter, but at the same time we must not unduly slow response to the high-frequency component of a genuine saccade.

"Midas Touch" Problem

The most naive approach to using eye position as an input might be to use it as a direct substitute for a mouse: changes in the user's line of gaze would cause the mouse cursor to move. This is an unworkable (and annoying) approach, because people are not accustomed to operating devices just by moving their eyes. They expect to be able to look at an item without having the look "mean" something. Normal visual perception requires that the eyes move about, scanning the scene before them. It is not desirable for each such move to initiate a computer command.

At first, it is empowering to be able simply to look at what you want and have it happen, rather than having to look at it (as you would anyway) and then point and click it with the mouse or otherwise issue a command. Before long, though, it becomes like the Midas Touch. Everywhere you look, another command is activated; you cannot look anywhere without issuing a command. The challenge in building a useful eye tracker interface is to avoid this Midas Touch problem. Ideally, the interface should act on the user's eye input when he wants it to and let him just look around when that's what he wants, but the two cases are impossible to distinguish in general. Instead, we investigate interaction techniques that address this problem in specific cases.

3. EXPERIENCE WITH EYE MOVEMENTS

Configuration

We use an Applied Science Laboratories corneal reflection eye tracker. The user sits at a conventional (government-issue) desk, with a Sun computer display, mouse, and keyboard, in a standard chair and office. The eye tracker camera/illuminator sits on the desk next to the monitor. Other than the illuminator box with its dim red glow, the overall setting is thus far just like that for an ordinary office computer user. In addition, the room lights are dimmed to keep the user's pupil from becoming too small. The eye tracker transmits the x and y coordinates for the user's visual line of gaze every $\frac{1}{60}$ second, on a serial port, to a Sun 4/260 computer. The Sun performs all further processing, filtering, fixation recognition, and some additional calibration. Software on the Sun parses the raw eye tracker data stream into

tokens that represent events meaningful to the user-computer dialogue. Our User Interface Management System [7] multiplexes these tokens with other inputs (such as mouse and keyboard) and processes them to implement the user interfaces under study.

Observation. The eye tracker is, strictly speaking, nonintrusive and does not touch the user in any way. Our setting is almost identical to that for a user of a conventional office computer. Nevertheless, we find it is difficult to ignore the eye tracker. It is noisy; the dimmed room lighting is unusual; the dull red light, while not annoying, is a constant reminder of the equipment; and, most significantly, the action of the servo-controlled mirror, which results in the red light following the slightest motions of user's head gives one the eerie feeling of being watched. One further wrinkle is that the eye tracker is designed for use in experiments, where there is a "subject" whose eye is tracked and an "experimenter" who monitors and adjusts the equipment. Operation by a single user playing both roles simultaneously is somewhat awkward because, as soon as you look at the eye tracker control panel to make an adjustment, your eye is no longer pointed where it should be for tracking.

Accuracy and Range

A user generally need not position his eye more accurately than the width of the fovea (about one degree) to see an object sharply. Finer accuracy from an eye tracker might be needed for studying the operation of the eye muscles but adds little for our purposes. The eye's normal jittering further limits the practical accuracy of eye tracking. It is possible to improve accuracy by averaging over a fixation, but not in a real-time interface.

Observation. Despite the servo-controlled mirror mechanism for following the user's head, we find that the steadier the user holds his head, the better the eye tracker works. We find that we can generally get two degrees accuracy quite easily, and sometimes can achieve one degree (or approximately 0.4″ on the screen at a 24″ viewing distance). The eye tracker should thus be viewed as having a resolution much coarser than that of a mouse or other typical devices, perhaps more like a traditional touch screen. A further problem is that the range over which the eye can be tracked with this equipment is fairly limited. In our configuration, it can barely cover the surface of a 19″ monitor at a 24″ viewing distance.

Using the Eye Tracker Data

Our approach to processing eye movement data is to partition the problem into two stages. First we process the raw data from the eye tracker in order to filter noise, recognize fixations, compensate for local calibration errors, and generally try to reconstruct the user's more conscious intentions from the available information. This processing stage converts the continuous, somewhat noisy stream of raw eye position reports into discrete tokens that we

claim more closely approximate the user's intentions in a higher-level user-computer dialogue. Then, we design generic interaction techniques based on these tokens as inputs.

Observation. Because eye movements are so different from conventional computer inputs, we achieve success with a philosophy that tries, as much as possible, to use natural eye movements as an implicit input, rather than to train a user to move the eyes in a particular way to operate the system. We try to think of eye position more as a piece of information available to the user-computer dialogue involving a variety of input devices than as the intentional actuation of an input device.

Local Calibration

The eye tracker calibration procedure produces a mapping that is applied uniformly to the whole screen. Ideally, no further calibration or adjustment is necessary. In practice, we found small calibration errors appear in portions of the screen, rather than systematically across it. We introduced an additional layer of calibration into the chain, outside of the eye tracker computer, which allows the user to make local modifications to the calibration, based on arbitrary points he inputs whenever he feels it would be helpful. If the user feels the eye tracker is not responding accurately in some area of the screen, he can at any point move the mouse cursor to that area, look at the cursor, and click a button.

Observation. Surprisingly, this had the effect of increasing the apparent response speed for object selection and other interaction techniques. The reason is that, if the calibration is slightly wrong in a local region and the user stares at a single target in that region, the eye tracker will report the eye position somewhere slightly outside the target. If he continues to stare at it, though, his eyes will in fact jitter around to a spot that the eye tracker will report as being on the target. The effect feels as though the system is responding too slowly, but it is a problem of local calibration. The local calibration procedure results in a marked improvement in the apparent responsiveness of the interface as well as an increase in the user's control over the system (since he can recalibrate when and where desired).

Fixation Recognition

After improving the calibration, we still observed what seemed like erratic behavior in the user interface, even when the user thought he was staring perfectly still. This was caused by both natural and artificial sources: the normal jittery motions of the eye during fixations as well as artifacts introduced when the eye tracker momentarily fails to obtain an adequate video image of the eye.

Figure 3 shows the type of data obtained from the eye tracker. It plots the x coordinate of the eye position output against time over a relatively jumpy three second period. (A plot of the y coordinate for the same period would show generally the same areas of smooth versus jumpy behavior, but differ-

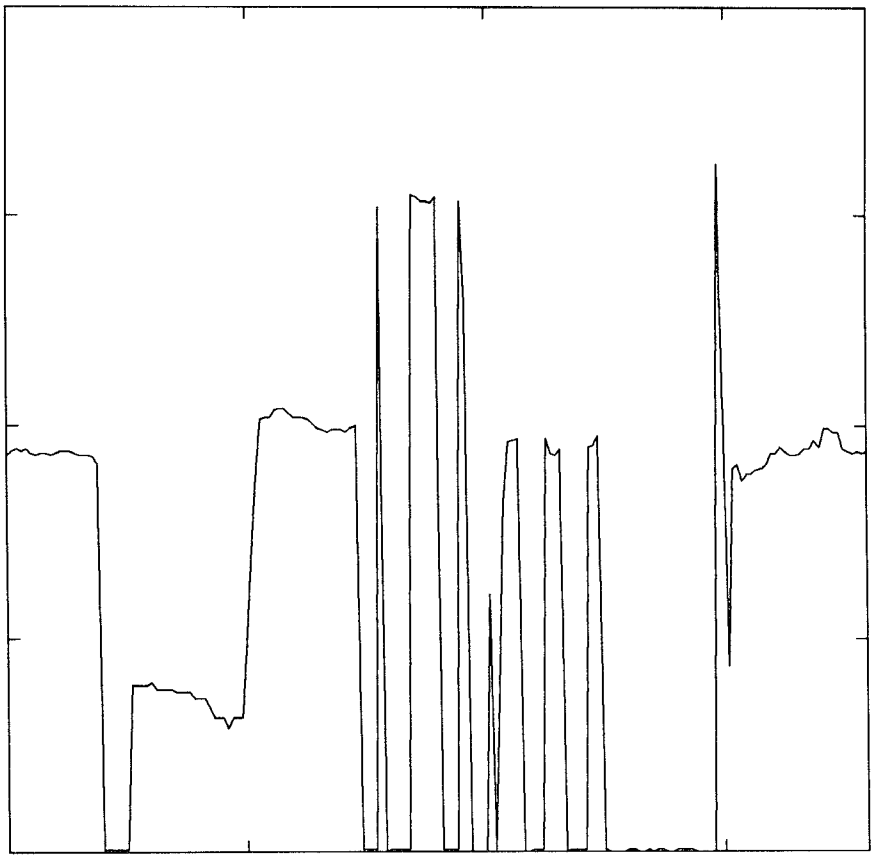

Fig. 3. Illustration of erratic nature of raw data from the eye tracker. The plot shows one coordinate of eye position versus time, over a somewhat worse-than-typical three second period.

ent absolute positions.) Zero values on the ordinate represent periods when the eye tracker could not locate the line of gaze, due either to eye tracker artifacts, such as glare in the video camera, lag in compensating for head motion, or failure of the processing algorithm, or by actual user actions, such as blinks or movements outside the range of the eye tracker. Unfortunately, the two cases are indistinguishable in the eye tracker output. During the period represented by Figure 3, the subject thought he was simply looking around at a few different points on a CRT screen. Buried in these data, thus, are a few relatively long gazes along with some motions to connect the gazes. Such raw data are quite unusable as input to a human-computer dialogue: while the noise and jumpiness do partly reflect the actual motion of the user's eye muscles, they do not reflect his intentions nor his impression of what his eyes were doing. The difference is attributable not only to the eye tracker artifacts but to the fact that much of the fine-grained behavior of the eye muscles is not intentional.

The problem is to extract from the noisy, jittery, error-filled stream of position reports produced by the eye tracker some "intentional" components of the eye motions, which make sense as tokens in a user-computer dialogue. Our first solution was to use a simple moving average filter to smooth the data. It improves performance during a fixation, but tends to dampen the sudden saccades that move the eye from one fixation to the next. Since one of the principal benefits we hope to obtain from eye motions as input is speed, damping them is counterproductive. Further, the resulting smoothed data do not correctly reflect the user's intentions. The user was not slowly gliding from one fixation to another; he was, in fact, fixating a spot and then jumping ballistically to a new fixation.

Instead, we return to the picture of a computer user's eye movements as a collection of jittery fixations connected by essentially instantaneous saccades. We start with an *a priori* model of such saccades and fixations and then attempt to recognize those events in the data stream. We then identify and quickly report the start and approximate position of each recognized fixation. We ignore any reports of eye position during saccades themselves, since they are difficult for the eye tracker to catch and their dynamics are not particularly meaningful to the user-computer dialogue.

Specifically, our algorithm, which is based on that used for retrospective analysis of eye tracker data and on the known properties of fixations and saccades, watches the input data for a sequence of 100 milliseconds during which the reported eye position remains within approximately 0.5 degrees. As soon as the 100 milliseconds have passed, it reports the start of a fixation and takes the mean of the set of data collected during the 100 milliseconds duration as the location of that fixation. A better estimate of the location of a fixation could be obtained by averaging over more eye tracker data, but this would mean a longer delay before the fixation position could be reported to the user interface software. Our algorithm implies a delay of 100 milliseconds before reporting the start of a fixation, and, in practice this delay is nearly undetectable to the user. Further eye positions within approximately one degree are assumed to represent continuations of the same fixation (rather than a saccade to a new one). To terminate a fixation, 50 milliseconds of data lying outside one degree of the current fixation must be received. Blinks or artifacts of up to 200 milliseconds may occur during a fixation without terminating it. (These occur when the eye tracker reports a "no position" code.) At first, blinks seemed to present a problem, since, obviously, we cannot obtain eye position data during a blink. However (equally obviously in retrospect), the screen need not respond to the eye during that blink period, since the user can't see it anyway.

After applying this algorithm, the noisy data shown in Figure 3 are found to comprise about 6 fixations, which more accurately reflects what the user thought he was doing (rather than what his eye muscles plus the eye tracking equipment actually did). Figure 4 shows the same data, with a horizontal line marking each recognized fixation at the time and location it would be reported.

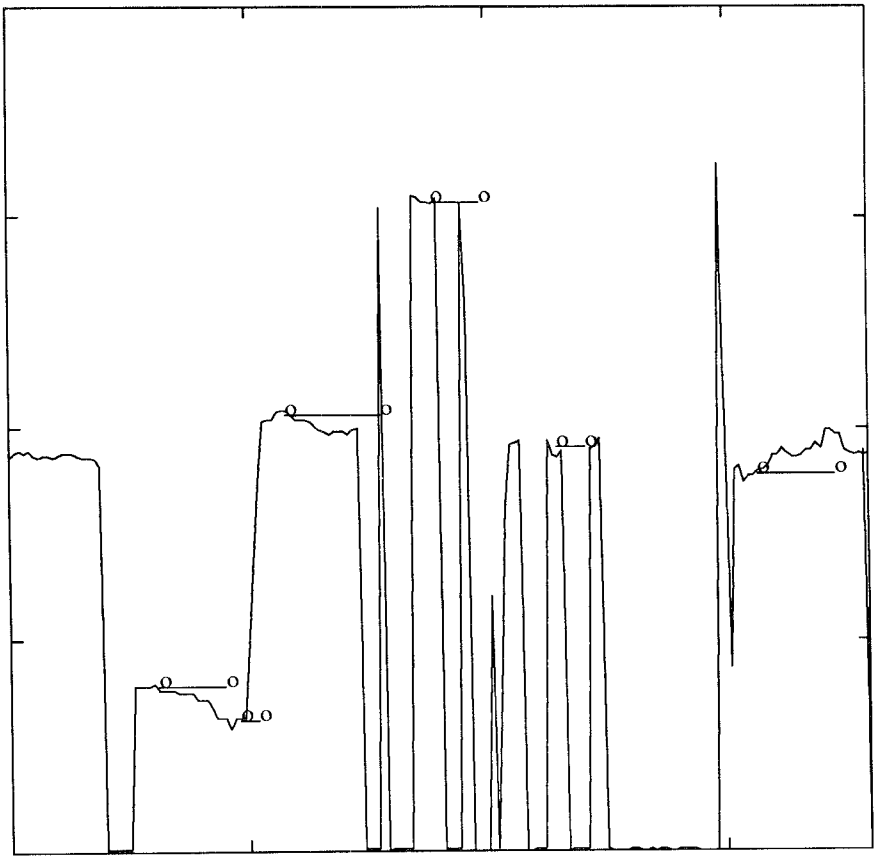

Fig. 4. Result of applying the fixation recognition algorithm to the data of Figure 3. A horizontal line beginning and ending with an **o** marks each fixation at the time and coordinate position it would be reported.

Observation. Applying the fixation recognition approach to the real-time data coming from the eye tracker yielded a significant improvement in the user-visible behavior of the interface. Filtering the data based on an *a priori* model of eye motion is an important step in transforming the raw eye tracker output into a user-computer dialogue.

User Interface Management System

In order to make the eye tracker data more tractable for use as input to an interactive user interface, we turn the output of the recognition algorithm into a stream of *tokens*. We report tokens for eye events considered meaningful to the user-computer dialogue, analogous to the way that raw input from a keyboard (shift key went down, letter *a* key went down, etc.) is turned into meaningful events (one ASCII upper case *A* was typed). We report tokens for

the start, continuation (every 50 milliseconds, in case the dialogue is waiting to respond to a fixation of a certain duration), and end of each detected fixation. Each such token is tagged with the actual fixation duration to date, so an interaction technique that expects a fixation of a particular length will not be skewed by delays in UIMS processing or by the delay inherent in the fixation recognition algorithm. In between fixations, we periodically report a nonfixation token indicating where the eye is, although our current interaction techniques ignore this token in preference to the more processed fixation tokens. A token is also reported whenever the eye tracker fails to determine eye position for 200 milliseconds and again when it resumes tracking. These tokens, having been processed by the algorithms described above, are suitable for use in a user-computer dialogue in the same way as tokens generated by mouse or keyboard events.

We then multiplex the eye tokens into the same stream with those generated by the mouse and keyboard and present the overall token stream as input to our User Interface Management System. The desired user interface is specified to the UIMS as a collection of concurrently executing interaction objects [7]. The operation of each such object is described by a state transition diagram that accepts the tokens as input. Each object can accept any combination of eye, mouse, and keyboard tokens, as specified in its own syntax diagram.

4. INTERACTION TECHNIQUES

An interaction technique is a way of using a physical input device to perform a generic task in a human-computer dialogue [12]. It represents an abstraction of some common class of interactive task, for example, choosing one of several objects shown on a display screen. This section describes the first few eye movement-based interaction techniques that we have implemented and our initial observations from using them.

Object Selection

The task here is to select one object from among several displayed on the screen, for example, one of several file icons on a desktop or, as shown in Figure 5, one of several ships on a map in a hypothetical "command and control" system. With a mouse, this is usually done by pointing at the object and then pressing a button. With the eye tracker, there is no natural counterpart of the button press. We reject using a blink for a signal because it detracts from the naturalness possible with an eye movement-based dialogue by requiring the user to think about when he or she blinks. We tested two alternatives. In one, the user looks at the desired object then presses a button on a keypad to indicate that the looked-at object is his choice. In Figure 5, the user has looked at ship "EF151" and caused it to be selected (for attribute display, described below). The second uses dwell time—if the user continues to look at the object for a sufficiently long time, it is selected without further operations. The two techniques are actually implemented simultaneously, where the button press is optional and can be used to avoid

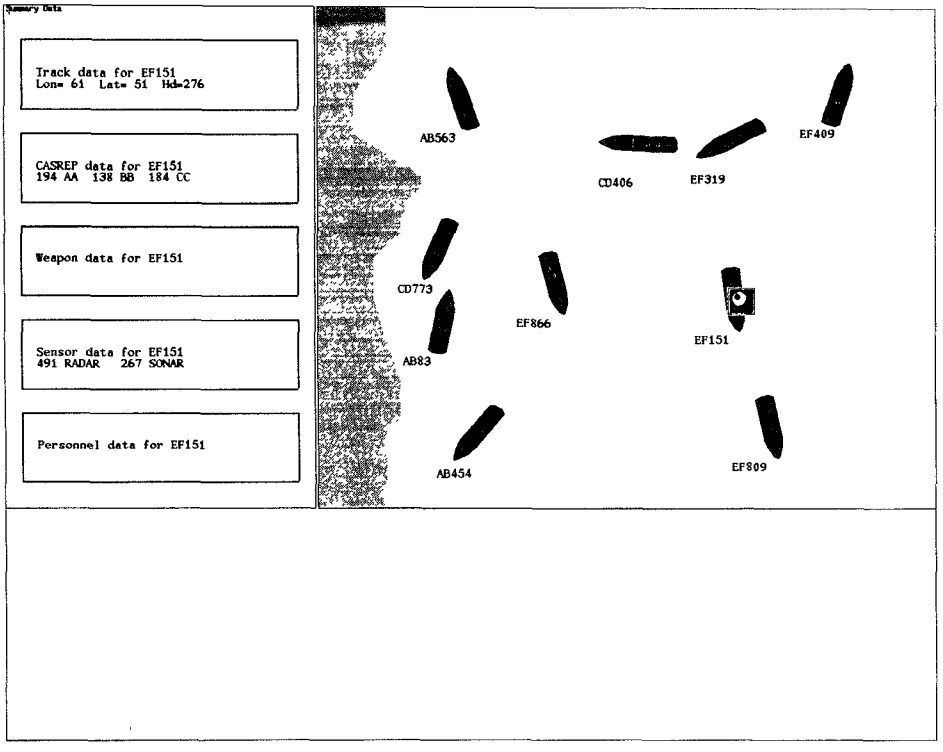

Fig 5. Display from eye tracker testbed, illustrating object selection technique. Whenever the user looks at a ship in the right window, the ship is selected and information about it is displayed in left window. The square eye icon at the right is used to show where the user's eye was pointing in these illustrations; it does not normally appear on the screen. The actual screen image uses light figures on a dark background to keep the pupil large.

waiting for the dwell time to expire, much as an optional menu accelerator key is used to avoid traversing a menu.

Observation. At first this seemed like a good combination. In practice, however, the dwell time approach is much more convenient. While a long dwell time might be used to ensure that an inadvertent selection will not be made by simply "looking around" on the display, this mitigates the speed advantage of using eye movements for input and also reduces the responsiveness of the interface. To reduce dwell time, we make a further distinction. If the result of selecting the wrong object can be undone trivially (selection of a wrong object followed by a selection of the right object causes no adverse effect—the second selection instantaneously overrides the first), then a very short dwell time can be used. For example, if selecting an object causes a display of information about that object to appear and the information display can be changed instantaneously, then the effect of selecting wrong objects is immediately undone as long as the user eventually reaches the right one. This approach, using a 150–250 milliseconds dwell time gives

excellent results. The lag between eye movement and system response (required to reach the dwell time) is hardly detectable to the user, yet long enough to accumulate sufficient data for our fixation recognition and processing. The subjective feeling is of a highly responsive system, almost as though the system is executing the user's intentions before he expresses them. For situations where selecting an object is more difficult to undo, button confirmation is used. We found no case where a long dwell time (over $\frac{3}{4}$ second) alone was useful, probably because it does not exploit natural eye movements (people do not normally fixate one spot for that long) and also creates the suspicion that the system has crashed. Pilot studies for an experiment that will compare response time for object selection by dwell time versus conventional selection by mouse pick, using the more abstract display shown in Figure 6, suggest a 30 percent decrease in time for the eye over the mouse, although the eye trials show more variability.

Continuous Attribute Display

A good use of this object selection interaction technique is for requesting further details or attributes of one of the objects on a display. Our approach is to provide a separate area of the display where such attributes are always shown. In Figure 5, the window on the right is a geographic display of ships, while the text window on the left shows some attributes of one of the ships, the one selected by the user's eye movement. The idea behind this is that the user can look around the ship window as desired. Whenever he looks over to the text window, he will always find there the attribute display for the last ship looked at—presumably the one he is interested in. (The ship remains selected when he looks away from the ship window to the text window.) However, if he simply looks at the ship window and never looks at the text area, he need not be concerned that his eye movements are causing commands in the text window. The text window is double-buffered, so that changes in its contents could hardly be seen unless the user were looking directly at it at the time it changed (which, of course, he is not—he must be looking at the ship window to effect a change).

Moving an Object

Another important interaction technique, particularly for direct manipulation systems, is moving an object on the display. We have experimented with two methods. Our initial notion was that, in a direct manipulation system, a mouse is typically used for two distinct operations—selecting an object to be manipulated and performing the manipulation. The two functions could be separated and each assigned to an appropriate input device. In particular, the selection could be performed by eye position, while the hand input device is devoted exclusively to the manipulations. We therefore implemented a technique whereby the eye selects an object (ship) to be manipulated (moved on the map, in this case) and then the mouse is used to move it. The eye selection is made precisely as in the previously-described interaction techniques. Then, the user grabs the mouse, presses a button, drags the mouse in

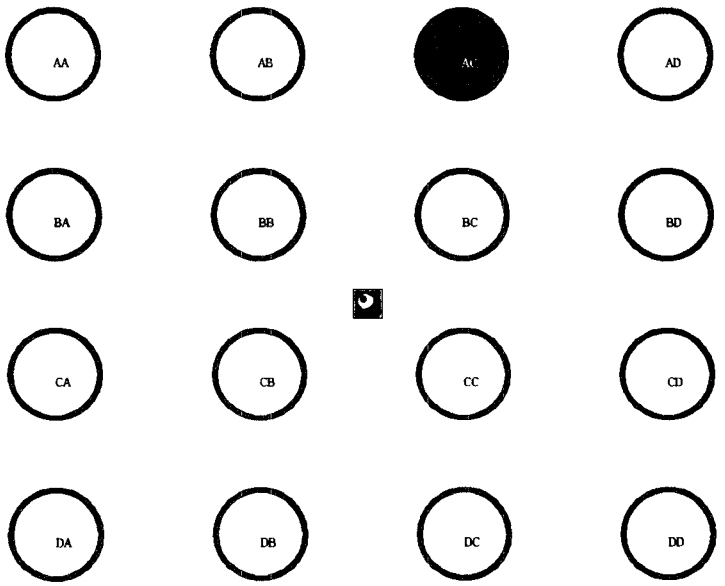

Fig. 6. Display for experimental study of the object selection interaction technique. Item "AC" near the upper right has just become highlighted, and the user must now select it (by eye or mouse).

the direction the object is to be moved, and releases the button. There is no visible mouse cursor in this scheme, and the mouse is used as a relative position device—it starts moving from wherever the eye-selected ship was. Our second approach used the eye to select and drag the ship, and a pushbutton to pick it up and put it down. The user selects a ship, then presses a button; while the button is depressed, the ship drags along with the user's eye. When it is released, the ship remains in its new position. Since the processing described previously is performed on the eye movements, the ship actually jumps to each fixation after about 100 milliseconds and then remains steadily there—despite actual eye jitter—until the next fixation.

Observation. Our initial guess was that the second method would be difficult to use: eye movements would be fine for selecting an object, but picking it up and having it jump around on the screen in response to eye movements would be annoying—a mouse would give more concrete control. Once again, our initial guess was not borne out. While the eye-to-select/mouse-to-drag method worked well, the user was quickly spoiled by the eye-only method. Once you begin to expect the system to know where you are looking, the mouse-to-drag operation seems awkward and slow. After looking at the desired ship and pressing the "pick up" button, the natural thing to do is to look at where you are planning to move the ship. At this point, you feel, "I'm looking right at the destination I want, why do I now have to go get the mouse to drag the ship over here?" With eye movements

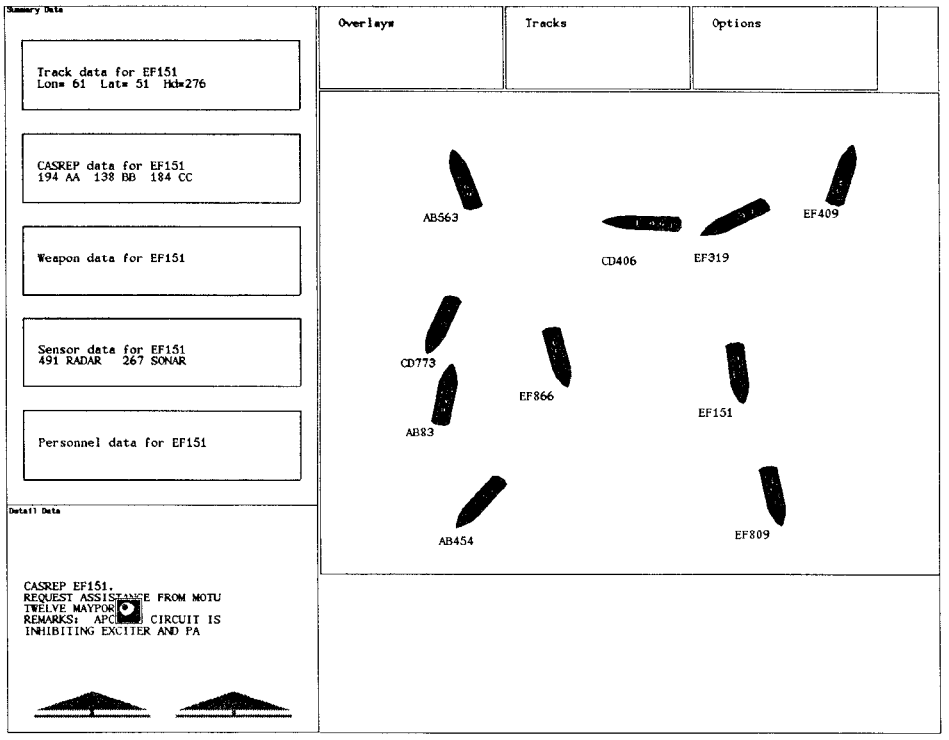

Fig. 7. Another display from the testbed, showing the scrolling text and other windows

processed to suppress jitter and respond only to recognized fixations, the motion of the dragging ship is reasonably smooth and predictable and yet appears subjectively instantaneous. It works best when the destination of the move is a recognizable feature on the screen (another ship, or a harbor on a map); when the destination is an arbitrary blank spot, it is more difficult to make your eye look at it, as the eye is always drawn to features.

Eye-Controlled Scrolling Text

A window of text is shown, but not all of the material to be displayed can fit. As shown at the bottom left of Figure 7, arrows appear below the last line of the text and above the first line, indicating that there is additional material not shown. If the user looks at an arrow, the text itself starts to scroll. Note, though, that it never scrolls when the user is actually reading the text (rather than looking at the arrow). The assumption is that, as soon as the text starts scrolling, the user's eye will be drawn to the moving display and away from the arrow, which will stop the scrolling. The user can thus read down to the end of the window, then, after he finishes reading the last line, look slightly below it, at the arrow, in order to retrieve the next part of the

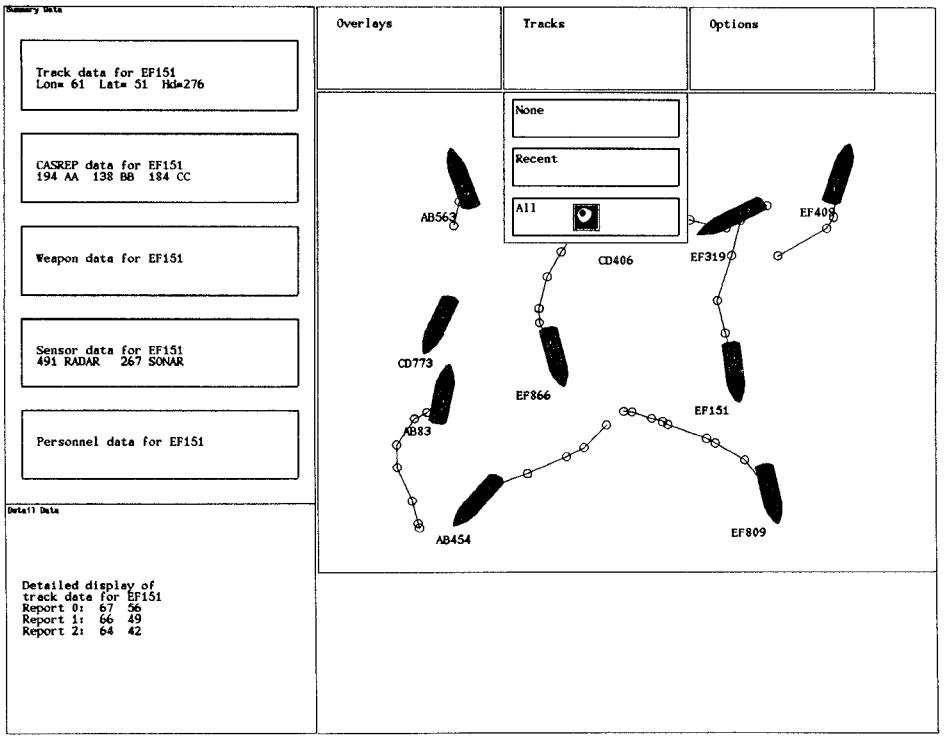

Fig. 8. Testbed display showing eye-controlled pull-down menu.

text. The arrow is visible above and/or below text display only when there is additional scrollable material in that direction.

Menu Commands

Another interaction technique is for choosing a command from a menu. Since pop-up menus inherently assume a button, we experimented with an eye-operated pull-down menu. In Figure 8, if the user looks at the header of a pull-down menu for a given dwell time (400 milliseconds), the body of the menu will appear on the screen. Next, he can look at the items shown on the menu. After a brief look at an item (100 milliseconds), it will be highlighted, but its command will not yet be executed. This allows the user time to examine the different items on the menu. If the user looks at one item for a much longer time (1 second), its command will be executed and the menu erased. Alternatively, once the item is highlighted, pressing a button will execute its command immediately and erase the menu. If the user looks outside the menu (for 600 milliseconds), the menu is erased without any command executed.

Observation. Our initial experience with this interaction technique suggests that the button is more convenient than the long dwell time for

executing a menu command. This is because the dwell time necessary before executing a command must be kept quite high, at least noticeably longer than the time required to read an unfamiliar item. This is longer than people normally fixate on one spot, so selecting such an item requires an unnatural sort of "stare." Pulling the menu down and selecting an item to be highlighted are both done very effectively with short dwell times, as with object selection.

Listener Window

In a window system, the user must designate the active or "listener" window, that is, the one that receives keyboard inputs. Current systems use an explicit mouse command to designate the active window; in some, the command is simply pointing, in others, it is pointing and clicking. Instead, we use eye position—the listener window is simply the one the user is looking at. A delay is built into the system, so that the user can look briefly at other windows without changing the listener window designation. Fine cursor motions within a window are still handled with the mouse, which gives an appropriate partition of tasks between eye tracker and mouse, analogous to that between speech and mouse used by Schmandt [13]. A possible extension to this approach is for each window to remember the location of the mouse cursor within it when the user last left that window. When the window is reactivated (by looking at it), the mouse cursor is restored to that remembered position.

5. OBSERVATIONS

Following Brooks' taxonomy [3], we present "observations," rather than more formal "findings" of our research at this point:

An eye tracker as an input device is far from "perfect," in the sense that a mouse or keyboard is, and that is caused both by the limitations of current equipment and, more importantly, by the nature of human eye movements. Accuracy obtainable is more similar to a traditional touch screen than a mouse, and the range can barely cover a single CRT display. The equipment, while nonintrusive and noncontacting, is still difficult to ignore. Nevertheless, it is perhaps amazing that this can be done at all; and, when the system is working well, it can give the powerful impression of responding to its user's intentions rather than his explicit inputs.

To achieve this, our overall approach in designing interaction techniques is, wherever possible, to obtain information from a user's *natural* eye movements while viewing the screen rather than requiring the user to make specific eye movements to actuate the system. We also found it important to search for and recognize fixations in the raw eye tracker data stream and construct our dialogue around these higher-level events.

In our initial interaction techniques, we observed the value of short dwell time eye-only object selection for cases where a wrong pick immediately followed by a correct pick is acceptable. For moving an object we found filtered eye movements surprisingly effective, even though a mouse initially

seemed more appropriate for this task. For menu commands, we found the eye alone appropriate for popping up a menu or tentatively choosing an item, but executing an item requires a button for confirmation rather than a long dwell time.

ACKNOWLEDGMENTS

I want to thank my colleagues, Robert Carter, Connie Heitmeyer, Preston Mullen, Linda Sibert, Stan Wilson, and Diane Zimmerman, for all kinds of help with this research.

REFERENCES

1. BOLT, R. A. Gaze-orchestrated dynamic windows. *Comput. Graph. 15*, 3 (Aug. 1981), 109–119.
2. BOLT, R. A. Eyes at the interface. In *Proceedings of the ACM Human Factors in Computer Systems Conference* (Gaithersburg, MD, Mar. 15–17, 1982), pp. 360–362.
3. BROOKS, F P Grasping reality through illusion-interactive graphics serving science. In *Proceedings of the ACM CHI'88 Human Factors in Computing Systems Conference* (Washington, D.C., May 15–19, 1988), Addison-Wesley/ACM Press, pp. 1–11.
4. CARD, S., ENGLISH, W., AND BURR, B. Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys for text selection on a CRT. *Ergonomics 21*, 8 (1978), 601–613.
5. GLENN, F. A., IAVECCHIA, H. P., ROSS, L. V., STOKES, J. M., WEILAND, W. J., WEISS, D., AND ZAKLAD, A. L. Eye-voice-controlled interface. In *Proceedings of the 30th Annual Meeting of the Human Factors Society* (Santa Monica, Calif., 1986), pp. 322–326.
6. HUTCHINSON, T. E., WHITE, K. P., MARTIN, W. N., REICHERT, K. C., AND FREY, L. A. Human-computer interaction using eye-gaze input. *IEEE Trans. Syst. Man and Cybern. 19*, 6 (Nov. 1989), 1527–1534.
7. JACOB, R. J. K. A specification language for direct manipulation user interfaces. *ACM Trans. Graph. 5*, 4 (1986), 283–317. Special Issue on User Interface Software.
8. JUST, M. A., AND CARPENTER, P. A. A theory of reading: From eye fixations to comprehension. *Psychological Rev. 87*, 4 (Jul. 1980), 329–354
9. LEVINE, J. L. An eye-controlled computer. IBM Thomas J. Watson Research Center, Res. Rep. RC-8857, Yorktown Heights, N.Y., 1981.
10. MERCHANT, J., MORRISSETTE, R., AND PORTERFIELD, J. L. Remote measurement of eye direction allowing subject motion over one cubic foot of space. *IEEE Trans. Biomed. Eng. BME-21*, 4 (Jul. 1974), 309–317.
11. MONTY, R. A., AND SENDERS, J. W. *Eye Movements and Psychological Processes*. Lawrence Erlbaum, Hillsdale, N.J., 1976.
12. MYERS, B. A. User-interface tools: Introduction and survey. *IEEE Softw. 6*, 1 (Jan. 1989), 15–23.
13. SCHMANDT, C , ACKERMAN, M. S., AND HINDUS, D. Augmenting a window system with speech input. *IEEE Comput. 23*, 8 (Aug. 1990), 50–56.
14. STARKER, I., AND BOLT, R. A. A gaze-responsive self-disclosing display. In *Proceedings of the ACM CHI'90 Human Factors in Computing Systems Conference* (Seattle, Wash., Apr. 1–5, 1990), Addison-Wesley/ACM Press, pp. 3–9.
15. WARE, C., AND MIKAELIAN, H. T. An evaluation of an eye tracker as a device for computer input. In *Proceedings of the ACM CHI + GI'87 Human Factors in Computing Systems Conference* (Toronto, Canada, Apr. 5–9, 1987), pp. 183–188.
16. YOUNG, L. R., AND SHEENA, D. Survey of eye movement recording methods. *Behav. Res. Meth. Instrument. 7*, 5 (1975), 397–429.