Eye Movement-Based Human-Computer Interaction Techniques: Toward Non-Command Interfaces

Robert J.K. Jacob

Human-Computer Interaction Lab
Naval Research Laboratory
Washington, D.C.

ABSTRACT

User-computer dialogues are typically one-sided, with the bandwidth from computer to user far greater than that from user to computer. The movement of a user's eyes can provide a convenient, natural, and high-bandwidth source of additional user input, to help redress this imbalance. We therefore investigate the introduction of eye movements as a computer input medium. Our emphasis is on the study of interaction techniques that incorporate eye movements into the user-computer dialogue in a convenient and natural way. This chapter describes research at NRL on developing such interaction techniques and the broader issues raised by non-command-based interaction styles. 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 movement-based interaction techniques that we have devised and implemented in our laboratory, reports our experiences and observations on them, and considers eye movement-based interaction as an exemplar of a new, more general class of non-command-based user-computer interaction.

I. INTRODUCTION

In searching for better interfaces between users and their computers, an additional mode of communication between the two parties would be of great use. The problem of human-computer interaction can be viewed as two powerful information processors (human and computer) attempting to communicate with each other via a narrow-bandwidth, highly constrained interface [25]. Faster, more natural, more convenient (and, particularly, more parallel, less sequential) means for users and computers to exchange information are needed to increase the useful bandwidth across that interface.

On the user's side, the constraints are in the nature of the communication organs and abilities with which humans are endowed; on the computer side, the only constraint is the range of devices and interaction techniques that we can invent and their performance. Current technology has been stronger in the computer-to-user direction than user-to-computer, hence today's user-computer dialogues are typically one-sided, with the bandwidth from the computer to the user far greater than that from user to computer. We are especially interested in input media that can help redress this imbalance by obtaining data from the user conveniently and rapidly. We therefore investigate the possibility of using the movements of a user's eyes to provide a high-bandwidth source of additional user input. While the technology for measuring a user's visual line of gaze (where he or she is looking in space) and reporting it 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. An interaction technique is a way of using a physical input device to perform a generic task in a human-computer dialogue [7].

Because eye movements are so different from conventional computer inputs, our basic approach to designing interaction techniques has been, wherever possible, to obtain information from the *natural* movements of the user's eye while viewing the display, rather than requiring the user to make specific *trained* eye movements to actuate the system. We therefore begin by studying the characteristics of natural eye movements and then attempt to recognize corresponding patterns in the raw data obtainable from the eye tracker, convert them into tokens with higher-level meaning, and then build dialogues based on the known characteristics of eye movements.

In addition, eye movement-based interaction techniques provide a useful exemplar of a new, non-command style of interaction. Some of the qualities that distinguish eye movement-based interaction from more conventional types of interaction are shared by other newly emerging styles of human-computer interaction that can collectively be characterized as ‘‘non-command-based.'' In a non-command-based dialogue, the user does not issue specific commands; instead, the computer passively observes the user and provides appropriate responses. Non-command-based interfaces will also have a significant effect on user interface software because of their emphasis on continuous, parallel input streams and real-time timing constraints, in contrast to conventional single-thread dialogues based on discrete tokens. We describe the simple user interface management system and user interface description language incorporated into our system and the more general requirements of user interface software for highly interactive, non-command styles of interaction.

Outline

This chapter begins by discussing the non-command interaction style. Then it focuses on eye movement-based interaction as an instance of this style. It introduces a taxonomy of the interaction metaphors pertinent to eye movements. It describes research at NRL on developing and studying eye movement-based 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 movement-based interaction techniques that we have devised and implemented in our laboratory, and reports our experiences and observations on them. Finally, the chapter returns to the theme of new interaction styles and attempts to identify and separate out the characteristics of non-command styles and to consider the impact of these styles on the future of user interface software.

II. NON-COMMAND INTERFACE STYLES

Eye movement-based interaction is one of several areas of current research in human-computer interaction in which a new interface style seems to be emerging. It represents a change in input from objects for the user to *actuate* by specific commands to passive equipment that simply *senses* parameters of the user's body. Jakob Nielsen describes this property as *non-command-based:*

> The fifth generation user interface paradigm seems to be centered around non-command-based dialogues. This term is a somewhat negative way of characterizing a new form of interaction but so far, the unifying concept does seem to be exactly the abandonment of the principle underlying all earlier paradigms: That a dialogue has to be controlled by specific and precise commands issued by the user and processed and replied to by the computer. The new interfaces are often not even dialogues in the traditional meaning of the word, even though they obviously can be analyzed as having some dialogue content at some level since they do involve the exchange of information between a user and a computer. The principles shown at CHI'90 which I am summarizing as being non-command-based interaction are eye tracking interfaces, artificial realities, play-along music accompaniment, and agents [19].

Previous interaction styles–batch, command line, menu, full-screen, natural language, and even current desktop or "WIMP" (window-icon-menu-pointer) styles–all await, receive, and respond to explicit commands from the user to the computer. In the non-command style, the computer passively monitors the user and responds as appropriate, rather than waiting for the user to issue specific commands. This distinction can be a subtle one, since any user action, even a non-voluntary one, could be viewed as a command, particularly from the point of view of the software designer. The key criterion should therefore be whether the user *thinks* he or she is issuing an explicit command. It is of course possible to control one's eye movements, facial expressions, or gestures voluntarily, but that misses the point of a non command-based interface; rather, it is supposed passively to observe, for example, the user's natural eye movements, and respond based on them. The essence of this style is thus its *non-intentional* quality. Following Rich's taxonomy of adaptive systems [13, 21], we can view this distinction as *expli-*

cit vs. *implicit* commands, thus non-command really means implicit commands.

This style of interface requires the invention of new interaction techniques that are helpful but do not annoy the user. Because the inputs are often non-intentional, they must be interpreted carefully to avoid annoying the user with unwanted responses to inadvertent actions. For eye movements, we have called this the "Midas Touch" problem, since the highly responsive interface is both a boon and a curse. Our investigation of eye movement-based interaction techniques, described in this chapter, provides an example of how these problems can be attacked.

III. PERSPECTIVES ON EYE MOVEMENT-BASED INTERACTION

As with other areas of user interface design, considerable leverage can be obtained by drawing analogies that use people's already-existing skills for operating in the natural environment and searching for ways to apply them to communicating with a computer. Direct manipulation interfaces have enjoyed great success, particularly with novice users, largely because they draw on analogies to existing human skills (pointing, grabbing, moving objects in physical space), rather than trained behaviors; and virtual realities offer the promise of usefully exploiting people's existing physical navigation and manipulation abilities. These notions are more difficult to extend to eye movement-based interaction, since few objects in the real world respond to people's eye movements. The principal exception is, of course, other people: they detect and respond to being looked at directly and, to a lesser and much less precise degree, to what else one may be looking at. In describing eye movement-based human-computer interaction we can draw two distinctions, as shown in Figure 1: one is in the nature of the user's eye movements and the other, in the nature of the responses. Each of these could be viewed as *natural* (that is, based on a corresponding real-world analogy) or *unnatural* (no real world counterpart):

- Within the world created by an eye movement-based interface, users could move their eyes to scan the scene, just as they would a real world scene, unaffected by the presence of eye tracking equipment (*natural* eye movement, on the eye movement axis of Figure 1). The alternative is to instruct users of the eye movement-based interface to move their eyes in particular ways, not necessarily those they would have employed if left to their own devices, in order to actuate the system (*unnatural* or learned eye movements).

- On the response axis, objects could respond to a user's eye movements in a natural way, that is, the object responds to the user's looking in the same way real objects do. As noted, there is a limited domain from which to draw such analogies in the real world. The alternative is unnatural response, where objects respond in ways not experienced in the real world.

This suggests the range of possible eye movement-based interaction techniques shown in Figure 1 (although the two axes are more like continua than sharp categorizations). The natural eye movement/natural response area is a difficult one, because it draws on a limited and subtle domain, principally how people respond to other people's gaze. Starker and Bolt [23] provide an excellent example of this mode, drawing on the analogy of a tour guide or host who estimates the visitor's interests by his or her gazes. In the work described in this chapter, we try to use natural (not trained) eye movements as input, but we provide responses unlike those in the real world. This is a compromise between full analogy to the real world and an entirely artificial interface. We present a display and allow the user to observe it with his or her normal scanning mechanisms, but such scans then induce responses from the computer not normally exhibited by real world objects. Most previous eye movement-based systems have used learned ("unnatural") eye movements for operation and thus, of necessity, unnatural responses. Much of that work has been aimed at disabled or hands-busy applications, where the cost of learning the required eye movements ("stare at this icon to activate the device") is repaid by the acquisition of an otherwise impossible new ability. However, we believe that the real benefits of eye movement interaction for the majority of users will be in its naturalness, fluidity, low cognitive load, and almost unconscious operation; these benefits are attenuated if unnatural, and thus quite conscious, eye movements are required. The remaining category in Figure 1, unnatural eye movement/natural response, is anomalous and has not been used in practice.

IV. CHARACTERISTICS OF EYE MOVEMENTS

In order to proceed with the design of effective eye movement-based human-computer interaction, we must first examine the characteristics of natural eye movements, with emphasis on those likely to be exhibited by a user in front of a conventional (non-eyetracking) computer console.

The Eye

The retina of the eye is not uniform. Rather, one small portion near its center contains many densely-packed receptors and thus permits sharp vision, while the rest of the retina permits only much blurrier vision. That central portion (the fovea) covers a field of view approximately one degree in diameter (the width of one word in a book held at normal reading distance or slightly less than the width of your thumb held at the end of your extended arm). Anything outside that area is seen only with ‘‘peripheral vision,’’ with 15 to 50 percent of the acuity of the fovea. It follows that, to see an object clearly, it is necessary to move the eye so that the object appears on the fovea. Conversely, because peripheral vision is so poor relative to foveal vision and the fovea so small, a person’s eye position gives a rather good indication (to within the one-degree width of the fovea) of what specific portion of the scene before the person is being examined.

Types of Eye Movements

Human eye movements can be grouped into several categories [10, 27].

- First, the principal method for moving the fovea to view a different portion of the visual scene is a sudden and rapid motion called a *saccade.* Saccades take approximately 30-120 milliseconds and traverse a range between 1 and 40 degrees of visual angle (15-20 degrees being most typical). Saccades are ballistic, that is, once begun, their trajectory and destination cannot be altered. Vision is suppressed (but not entirely prevented) during a saccade. There is a 100-300 ms. delay between the onset of a stimulus that might attract a saccade (e.g., an object appearing in peripheral vision) and the saccade itself. There is also a 200 ms. refractory period after one saccade before it is possible to make another one. Typically, a saccade is followed by a 200-600 ms. period of relative stability, called a *fixation*, during which an object can be viewed. The purpose of a saccade appears to be to get an object that lies somewhere in the visual field onto one's fovea for sharp viewing. Since the saccade is ballistic, such an object must be selected before the saccade is begun; peripheral vision must therefore be the means for selecting the target of each saccade.
- During a fixation, the eye does not remain still. Several types of small, jittery motions occur, generally less than one degree in size. There is a sequence of a slow drift followed by a sudden, tiny saccade-like jump to correct the effect of the drift (a microsaccade). Superimposed on these is a high-frequency tremor, like the noise seen in an imperfect servomechanism attempting to hold a fixed position.
- Another type of eye movement occurs only in response to a moving object in the visual field. This is a pursuit motion, much slower than a saccade and in synchrony with the moving object being viewed. Smooth pursuit motions cannot be induced

voluntarily; they require a moving stimulus.

- Yet another type of movement, called *nystagmus*, can occur in response to motions of the head. This is a pattern of smooth motion to follow an object (as the head motion causes it to move across the visual field), followed by a rapid motion in the opposite direction to select another object (as the original object moves too far away to keep in view). It can be induced by acceleration detected by the inner ear canals, as when a person spins his or her head around or twirls rapidly, and also by viewing a moving, repetitive pattern.

- The eyes also move relative to one another, to point slightly toward each other when viewing a near object or more parallel for a distant object. Finally, they exhibit a small rotation around an axis extending from the fovea to the pupil, depending on neck angle and other factors.

Thus the eye is rarely entirely still, even when viewing a static display. It constantly moves and fixates different portions of the visual field; it makes small, jittery motions even during a fixation; and it seldom remains in one fixation for long. Visual perception of a static scene appears to require the artificially induced changes caused by moving the eye around the scene. In fact, an image that is artificially fixed on the retina (every time the eye moves, the target immediately moves precisely the same amount) will appear to fade from view after a few seconds [20]. The large and small motions the eye normally makes prevent this fading from occurring outside the laboratory.

Implications

The overall picture of eye movements for a user sitting in front of a computer is, then, a collection of steady (but slightly jittery) fixations connected by sudden, rapid saccades. 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 wildly about the screen.

V. METHODS FOR MEASURING EYE MOVEMENTS

What to Measure

For human-computer dialogues, we wish to measure *visual line of gaze,* rather than simply the position of the eye in space or the relative motion of the eye within the head. Visual line of gaze is a line radiating forward in space from the eye; the user is looking at something along that line. To illustrate the difference, suppose an eye-tracking instrument detected a small lateral motion of the pupil. It could mean either that the user’s head moved in space (and his or her eye is still looking at nearly the same point) or that the eye rotated with respect to the head (causing a large change in where the eye is looking). We need to measure where the eye is pointing in space; not all eye tracking techniques do this. We do not normally measure how far out along the visual line of gaze the user is focusing (i.e., accommodation), but when viewing a two-dimensional surface like a computer console, it will be easy to deduce. Since both eyes generally point together, it is customary to track only one eye.

Electronic Methods

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 (i.e., AC electrode measurements) than absolute position (which requires DC measurements). It can cover a wide range of eye movements, but gives poor accuracy (particularly in absolute position). It is principally useful for diagnosing neurological problems revealed by eye movement patterns. Further details on this and the other eye tracking methods discussed here can be found in [27].

Mechanical Methods

Perhaps the least user-friendly approach uses a non-slipping contact lens ground to fit precisely over the corneal bulge. A slight suction is applied between the lens and the eye to hold it in place. The contact lens then has either a small mechanical lever, magnetic coil, or mirror attached for tracking. This method is extremely accurate, particularly for investigation of tiny eye movements, but practical only for laboratory studies. It is very awkward and uncomfortable, covers only a limited range, and interferes with blinking.

Optical/Video Methods – Single Point

More practical methods use remote imaging of some visible feature located on the eye, such as the boundary between the sclera (white portion of the front of the eye) and iris (colored portion)– this boundary is only partially visible at any one time, the outline of the pupil (works best for subjects with light-colored eyes or else the pupil can be illuminated so it appears lighter than the iris regardless of eye color), or the reflection off the front of the cornea of a collimated light beam shone at the eye. Any of these can then be used with photographic or video recording (for retrospective analysis) or with real-time video processing. They all require the head to be held absolutely stationary to be sure that any movement detected represents movement of the eye, rather than the head moving in space; a bite board is customarily used.

Optical/Video Methods – Two Point

However, by simultaneously tracking two features of the eye that move differentially with respect to one another as the line of gaze changes, it is possible to distinguish head movements (the two features move together) from eye movements (the two move with respect to one another). The head no longer need be rigidly fixed, although it must stay within camera range (which is quite small, due to the extreme telephoto lens required). Both the corneal reflection (from the light shining on the eye) *and* outline of the pupil (illuminated by the same light) are tracked. Infrared light is used, which is not disturbing to the subject. Then absolute visual line of gaze is computed from the relationship between the two tracked points. Temporal resolution is limited to the video frame rate (in particular, it cannot generally capture the dynamics of a saccade).

A related method used in the SRI eye tracker [5] tracks the corneal reflection plus the fourth Purkinje image (reflection from rear of lens); the latter is dim, so a bright illumination of the eye is needed. Reflections are captured by a photocell, which drives a servo-controlled mirror with an analog signal, avoiding the need for discrete sampling. Hence this method is not limited by video frame rate. The technique is accurate, fast, but very delicate to operate; it can also measure accommodation (focus distance).

Implications

While there are many approaches to measuring eye movements, most are more suitable for laboratory experiments than as an adjunct to normal computer use. The most reasonable method is the corneal reflection-plus-pupil outline approach, since nothing contacts the subject and the device permits his or her head to remain unclamped. In fact the eye tracker sits several feet away from the subject. Head motion is restricted only to the extent necessary to keep the pupil of the eye within view of the tracking camera. The camera is panned and focussed by a servomechanism that attempts to follow the eye as the subject’s head moves. The result is that the subject can move within approximately one cubic foot of space without losing contact with the eye tracker. Attached to the camera is an infrared illuminator that lights up the pupil (so that it is a bright circle against the dark iris) and also creates the corneal reflection; because the light is infrared, it is barely visible to the subject. With this method, the video image of the pupil is then analyzed to identify a large, bright circle (pupil) and a still brighter dot (corneal reflection) and compute the center of each; line of gaze is determined from these two points. This type of equipment is manufactured commercially; in our laboratory, we use an Applied

Science Laboratories (Waltham, Mass.) Model 3250R corneal reflection eye tracker [17, 27]. Figure 3 shows the components of this type of eye tracker.

VI. PREVIOUS WORK

While the 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 using eye movement data as a tool for studying motor and cognitive processes [14, 18]. Such work involves recording the eye movements and subsequently analyzing them; the user's eye movements do not have any effect on the computer interface while it is in operation.

For use as a component of a user interface, the eye movement data must be obtained in real time and used in some way that has an immediate effect on the dialogue. This situation has been studied most often for disabled (quadriplegic) users, who can use only their eyes for input. (e.g., [11, 15, 16] report work for which the primary focus was disabled users). Because all other user-computer communication modes are unavailable, the resulting interfaces are rather slow and tricky to use for non-disabled people, but, of course, a tremendous boon to their intended users.

One other case in which real-time eye movement data has been used in an interface is to create the illusion of a large, ultra-high resolution display in a flight simulator [24]. With this approach, the portion of the display that is currently being viewed is depicted with high resolution, while the larger surrounding area (visible only in peripheral vision) is depicted in lower resolution. Here, however, the eye movements are used essentially to simulate a better display device; the basic user-computer dialogue is not altered by the eye movements.

Our interest is, in contrast, in user-computer dialogues that combine real-time eye movement data with other, more conventional modes of user-computer communication. A relatively small amount of work has focussed on this particular problem. Richard Bolt did some of the earliest work and demonstrated several innovative uses of eye movements [1, 2, 23]. Floyd Glenn [8] used eye movements for several tracking tasks involving moving targets. Ware and Mikaelian [26] 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. The 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.

VII. PROBLEMS IN USING EYE MOVEMENTS IN A HUMAN-COMPUTER DIALOGUE

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 directly cause the mouse cursor to move. This turns out to be an unworkable (and annoying) design. There are two culprits for why direct substitution of an eye tracker for a mouse is not possible. The first is the eye itself, the jerky way it moves and the fact that it rarely sits still, even when its owner thinks he or she is looking steadily at a single object; the other is the instability of the available eye tracking hardware. There are significant differences between a manual input source like the mouse and eye position; some are advantages and some, disadvantages; they must all be considered in designing eye movement-based interaction techniques:

- First, as Ware and Mikaelian [26] observed, eye movement input is faster than other current input media. Before the user operates any mechanical pointing device, he or she usually looks at the destination to which he wishes to move. Thus the eye movement is available as an indication of the user's goal before he or she could actuate any other input device.

- Second, it is easy to operate. No training or particular coordination is required of normal users for them to be able to cause their eyes to look at an object; and the control-to-display relationship for this device is already established in the brain.
- The eye is, of course, much more than a high speed cursor positioning tool. Unlike any other input device, an eye tracker also tells where the user's interest is focussed. By the very act of pointing with this device, the user changes his or her focus of attention; and every change of focus is available as a pointing command to the computer. A mouse input tells the system simply that the user intentionally picked up the mouse and pointed it at something. An eye tracker input could be interpreted in the same way (the user intentionally pointed his or her eye at something, because he was trained to operate this system that way). But it can also be interpreted as an indication of what the user is currently paying attention to, without any explicit input action on his or her part.
- This same quality is the prime drawback of the eye as a computer input device. Moving one's eyes is often an almost subconscious act. Unlike a mouse, it is relatively difficult to control eye position consciously and precisely at all times. The eyes continually dart from spot to spot, and it is not desirable for each such move to initiate a computer command.
- Similarly, unlike a mouse, eye movements are always ‘‘on.’’ There is no natural way to indicate when to engage the input device, as there is with grasping or releasing the mouse. Closing the eyes is rejected for obvious reasons– even with eye-tracking as input, the principal function of the eyes in the user-computer dialogue is for communication *to* the user. Using blinks as a signal is unsatisfactory because it detracts from the naturalness possible with an eye movement-based dialogue by requiring the user to think about when to blink.

- Also, in comparison to a mouse, eye tracking lacks an analogue of the integral buttons most mice have. Using blinks or eye closings for this purpose is rejected for the reason mentioned.
- Finally, the eye tracking equipment is far less stable and accurate than most manual input devices.

‘‘Midas Touch’’ Problem

The problem with a simple implementation of an eye tracker interface is that people are not accustomed to operating devices simply by moving their eyes. They expect to be able to look at an item without having the look ‘‘mean’’ something. 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. 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 or she wants it to and let the user 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.

Jitter of Eye

During a fixation, a user generally thinks he or she 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, ideally, it should correspond to what the user *thinks* he or she 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.

Multiple ‘‘Fixations’’ in a Single ‘‘Gaze’’

A user may view a single object with a sequence of several fixations, all in the general area of the object. Since they are distinct fixations, separated by measurable saccades larger than the jitter mentioned above, they would be reported as individual fixations. Once again, if the user thinks he or she is looking at a single object, the user interface ought to treat the eye tracker data as if there were one event, not several. Therefore, following the approach of Just and Carpenter [14], if the user makes several fixations near the same screen object, connected by small saccades, we group them together into a single ‘‘gaze.’’ Further dialogue processing is performed in terms of these gazes, rather than fixations, since the former should be more indicative of the user’s intentions.

Instability in Eye Tracking Equipment

During operation of the eye tracker, there are often moments when the eye position is not available—the eye tracker fails to obtain an adequate video image of the eye for one or more frames. This could mean that the user blinked or moved his or her head outside the tracked region; if so, such information could be passed to the user interface. However, it could also mean simply that there was a spurious reflection in the video camera or any of a variety of other momentary artifacts. The two cases may not be distinguishable; hence, it is not clear how the user interface should respond to brief periods during which the eye tracker reports no position. The user may indeed have looked away, but he or she may also think he is looking right at some target on the screen, and the system is failing to respond.

Visual Feedback to User

One obvious question is whether the system should provide a screen cursor that follows the user’s eye position (as is done for mice and other conventional devices). If the eye tracker were perfect, the image of such a cursor would become stationary on the user’s retina and thus disappear from perception. In fact, few eye trackers can track small, high-frequency motions rapidly or precisely enough for this to be a problem, but it does illustrate the subtlety of the design issues.

A more immediate problem is that an eye-following cursor will tend to move around and thus attract the user’s attention. Yet it is perhaps the *least* informative aspect of the display (since it tells you where you are already looking). Further, if there is any systematic calibration error, the cursor will be slightly offset from where the user is actually looking, causing the user’s eye to be drawn to the cursor, which will further displace the cursor, creating a positive feedback loop. This is indeed a practical problem, and we often observe it.

Finally, if the calibration and response speed of the eye tracker were perfect, feedback would not be necessary, since a person knows exactly where he or she is looking (unlike the situation with a mouse cursor, which helps one visualize the relationship between mouse positions and points on the screen).

Implications

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 (described below) that are claimed to approximate more closely the user’s intentions in a higher-level user-computer dialogue. In doing so, jitter during fixations is smoothed, fixations are grouped into gazes, and brief eye tracker artifacts are removed.

Next, we design generic interaction techniques based on these tokens as inputs. Because eye movements are so different from conventional computer inputs, we achieve best results 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 address the ‘‘Midas Touch’’ problem by trying 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 the principal input device.

VIII. EXPERIENCE WITH EYE MOVEMENTS

Configuration

As noted, we use an Applied Science Laboratories eye tracker in our laboratory. The user sits at a conventional (government-issue) desk, with a 16" 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 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, closely modeled after that described in [12], multiplexes these tokens with other inputs (such as mouse and keyboard) and processes them to implement the user interfaces under study.

The eye tracker is, strictly speaking, non-intrusive 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 or her 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.

Despite the servo-controlled mirror mechanism for following the user's head, we find that the steadier the user holds his or her 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" or 40 pixels 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 most other pointing devices, perhaps more like a traditional touch screen. An additional problem is that the range over which the eye can be tracked with this equipment is fairly limited. In our configuration, it cannot quite cover the surface of a 19" monitor at a 24" viewing distance.

Local Calibration

Our first step in processing eye tracker data was to introduce an additional layer of calibration into the chain. 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 or she inputs whenever he feels it would be helpful. The procedure is that, if the user feels the eye tracker is not responding accurately in some area of the screen, he or she moves the mouse cursor to that area, looks at the cursor, and clicks a button. That introduces an offset, which warps future eye tracker reports in the vicinity of the given point, i.e., all reports nearer to that point than to the next-nearest local calibration point. (We found this gave better results than smoothly interpolating the local calibration offsets.) The user can do this at any time and in any position, as needed.

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 the user continues to stare at it, though, his or her 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 the user can re-calibrate 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 or she 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 4 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 vs. jumpy behavior, but different 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 4, this 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 analyzing previously-recorded files of raw eye movement 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 ms. have passed, it reports the start of a fixation and takes the mean of the 100 ms. worth of data 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 ms. 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 ms. of data lying outside one degree of the current fixation must be received. Blinks or artifacts of up to 200 ms. 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 4 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 5 shows the same data, with a horizontal line marking each recognized fixation at the time and location it would be reported.

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.

Re-assignment of Off-target Fixations

The processing steps described thus far are open-loop in the sense that eye tracker data are translated into recognized fixations at specific screen locations without reference to what is displayed on the screen. The next processing step is applied to fixations that lie outside the boundaries of the objects displayed on the screen. This step uses knowledge of what is actually on the screen, and serves further to compensate for small inaccuracies in the eye tracker data. It allows a fixation that is near, but not directly on, an eye-selectable screen object to be accepted. Given a list of currently displayed objects and their screen extents, the algorithm will reposition a fixation that lies outside any object, provided it is "reasonably" close to one object and "reasonably" further from all other such objects (i.e., not halfway between two objects, which would lead to unstable behavior). It is important that this procedure is applied only to fixations detected by the recognition algorithm, not to individual raw eye tracker position reports. The result of this step is to improve performance in areas of the screen at which the eye tracker calibration is imperfect; without increasing false selections in areas where the calibration is good, since fixations in those areas fall directly on their targets and would not activate this processing step.

IX. 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 ms., 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 processing by the UIMS (user interface management system) or by the delay inherent in the fixation recognition algorithm. Between fixations, we periodically report a non-fixation token indicating where the eye is, although our current interaction techniques ignore this token in preference to the fixation tokens, which are more filtered. A token is also reported whenever the eye tracker fails to determine eye position for 200 ms. and again when it resumes tracking. In addition, tokens are generated whenever a new fixation enters or exits a monitored region, just as is done for the mouse. Note that jitter during a single fixation will never cause such an enter or exit token, though, since the nominal position of a fixation is determined at the start of a fixation and never changes during the fixation. 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 [12]. The desired user interface is specified to the UIMS as a collection of relatively simple individual dialogues, represented by separate *interaction objects,* which comprise the user interface description language (UIDL). They are connected by an executive that activates and suspends them with retained state, like coroutines.

A typical object might be a screen button, scroll bar, text field, or eye-selectable graphic object. Since, at the level of individual objects, each such object conducts only a single-thread dialogue, with all inputs serialized and with a remembered state whenever the individual dialogue is interrupted by that of another interaction object, the operation of each interaction object is conveniently specified as a simple single-thread 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, and provides a standard method that the executive can call to offer it an input token and traverse its diagram. Each interaction object is also capable of redrawing itself upon command (it contains the needed state information or else contains calls to the access functions that will obtain such from its domain object). An interaction object can have different screen extents for purposes of re-drawing, accepting mouse tokens, and accepting eye tokens.

A standard executive is then defined for the outer dialogue loop. It operates by collecting all of the state diagrams of the interaction objects and executing them as a collection of coroutines, assigning input tokens to them and arbitrating among them as they proceed. Whenever the currently-active dialogue receives a token it cannot accept, the executive causes it to relinquish control by coroutine call to whatever dialogue can, given its current state, accept it. If none can, executive discards the token and proceeds.

For example, each ship (see Figure 6) is a separate interaction object (but all are of the same class, **Ship**). An additional lower-level interaction object (**Gazer**) is provided to perform the translation of fixations into gazes, as described above. That is, every interaction object such as **Ship** also has a **Gazer** interaction object associated with it. The **Gazer** accepts fixations on (or near, according to the criteria described above, and by means of a class variable shared by all the active **Gazer**s) its parent object and then combines such consecutive fixations into a single gaze token, which it sends to its parent object (the **Ship**). Figure 7 shows the syntax diagram for **Gazer**; it accepts the tokens generated by the fixation recognition algorithm (**EYEFIXSTART**, **EYEFIXCONT**, and **EYEFIXEND**), tests whether they lie within its extent or else meet the criteria for off-target fixations described above (implemented in the call to *IsMine*), accumulates them into gazes, and sends gaze tokens (**EYEGAZESTART**, **EYEGAZECONT**, and **EYEGAZEEND**) directly to its parent object. The **Ship** interaction object syntax then need only accept and respond to the gaze tokens sent by its **Gazer**. Figure 7 also shows the portion of the **Ship** interaction object syntax diagram concerned with selecting a ship by looking at it for a given dwell time (for clarity the syntax for dragging and other operations described below is not shown in the figure; also not shown are the tokens that the selected ship sends to the other ships to deselect the previously-selected ship, if any). When a user operation upon a ship causes a semantic-level consequence (e.g., moving a ship changes the track data), the **Ship** interaction object calls its parent, an application domain object, to do the work. Although the syntax may seem complicated as described here, it is well matched to the natural saccades and fixations of the eye.

X. INTERACTION TECHNIQUES

Interaction techniques provide a useful focus for this type of research because they are specific, yet not bound to a single application. An interaction technique represents an abstraction of some common class of interactive task, for example, choosing one of several objects shown on a display screen. Research in this area studies the primitive elements of human-computer dialogues, which apply across a wide variety of individual applications. The goal is to add new, high-bandwidth methods to the available store of input/output devices, interaction techniques, and generic dialogue components. Mockups of such techniques are then studied by measuring their properties, and attempts are made to determine their composition rules. This section describes the first few eye movement-based interaction techniques that we have implemented and our initial observations from using them.

Object Selection

This task 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 6, 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. As noted, we rejected using a blink and instead tested two alternatives. In one, the user looks at the desired object then presses a button on a keypad to indicate his or her choice. In Figure 6, the user has looked at ship ‘‘EF151’’ and caused it to be selected (for attribute display, described below). The second alternative 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 waiting for the dwell time to expire, much as an optional menu accelerator key is used to avoid traversing a menu. The idea is that the user can trade between speed and a free hand: if the user needs speed and can push the button he or she need not be delayed by eye tracker dwell time; if the user does not need maximum speed, then object selection reverts to the more passive eye-only mode using dwell time.

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 ms. 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 or she expresses them. For situations where selecting an object is more difficult to undo, button confirmation is used rather than a longer dwell time. We found no case where a long dwell time (over 3/4 second) alone was useful, probably because that is not a natural eye movement (people do not normally fixate one spot for that long) and also creates the suspicion that the system has crashed.

As described in the section on calibration, our initial implementation of this interaction technique performed very poorly. Even when we reduced the dwell time parameter drastically, it seemed too high. The culprit turned out to be small discrepancies between where the user was looking and what the eye tracker reported. Eliminating these errors improved the apparent responsiveness of the object selection technique. Further resistance to calibration errors is provided by an algorithm that accepts fixations outside a selectable object, provided they are fairly close to it and are substantially closer to it than to any other selectable objects. The processing of raw eye position data into fixations and then grouping them into gazes (described above) is also applied to this interaction technique.

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 6, 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, as selected by the user's eye movement. The idea is that the user can look around the ship window as desired. Whenever the user looks over to the text window, he or she will find the attribute display for the last ship looked at–presumably the one the user is interested in. (The ship remains selected when the user looks away from the ship window to the text window.) However, if the user simply looks at the ship window and never looks at the text area, he or she 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 are too subtle to be seen by peripheral vision. To notice the change, the user would have to be looking directly at the text window at the time it changed (which, of course, he or she is not–he must be looking at the ship window to effect a change). At present, the eye-selected ship also changes color slightly to confirm to the user that it is the one whose attributes are being displayed. This is largely to compensate for imperfections in the eye tracker, and our intention is ultimately to remove it, so the main window would truly be unaffected by eye movements within it.

Moving an Object

Another important interaction technique, particularly for direct manipulation systems, is moving an object on the display. We 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, and the hand input device 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 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. (Since the processing described previously is performed on the eye movements, the ship actually jumps to each fixation after about 100 ms. and then remains steadily there– despite actual eye jitter– until the next fixation.) When the button is released, the ship is left in its new position.

Our initial guess was that the second method would be too unpredictable; 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 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 8, a row of arrows appears 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 the arrows, 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 arrows). 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 arrows, which will stop the scrolling. The user can thus read down to end of the window, then, after he or she finishes reading the last line, look slightly below it, at the arrows, in order to retrieve the next part of the text. The arrows are visible above and/or below text display only when there is additional scrollable material in that direction.

Menu Commands

Another interaction technique is choosing a command from a menu. Since pop-up menus inherently assume a button, we used a pull-down menu. In Figure 9, if the user looks at the header of a pull-down menu for a given dwell time (400 ms.), the body of the menu will appear on the screen. Next, the user can look at the items shown on the menu. After a brief look at an item (100 ms.), 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 sec.), 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 ms.), the menu is erased without any command executed.

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 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 [22]. 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. This method works well in practice because the resolution required for selecting a window on a typical display is fairly coarse. It is convenient and fast, but the difficulty of using the eye tracker in a natural setting and posture for an extended time has made it difficult to frame a good comparison to a conventional window manager in everyday use.

XI. TOWARDS AND BEYOND NON-COMMAND INTERFACES

Finally, we return to the broader question of the nature of future human-computer interaction styles. Eye movement-based interaction provides an example of several of the characteristics – as well as the problems – of what seems to be an emerging new user-computer interaction style. The new style, which combines the non-command attribute with other somewhat correlated characteristics, is seen most dramatically in virtual reality interfaces, but its characteristics are common to a more general class of rich user-computer environments, such as new types of games, musical accompaniment systems, interactive entertainment media, as well as eye movement-based interfaces. They all share a higher degree of interactivity than previous interfaces – continuous input/output exchanges occurring in parallel, rather than one single-thread dialogue. Most also go a step further away from the traditional dialogue toward a more subtle, implicit interchange based on passive monitoring of the user's actions rather than explicit commands. The concepts behind these emerging new interface styles can be better understood by decomposing them into two main attributes, suggested in Figure 11. The non-command-based quality, which was described earlier, is just one of the two.

Command Style

One attribute is the change from explicit to implicit commands, the non-command-based interaction style. As mentioned earlier, non-command can be viewed more precisely as implicit command (in contrast to explicit). An intermediate point can be added along this axis. It still uses explicit commands, but is distinguished by whether interpretation of the user's command is entirely within the user's control or processed in a sufficiently complex way that the user can't easily predict its outcome. This arises in adaptive systems, which interpret a user's commands based on models of his or her state of knowledge and inferred goals and try to correct apparent user errors. The user issues explicit commands, but the precise consequence of each is out of his or her effective control.

Interactivity

The other attribute was initially christened non-WIMP [9], later refined to *highly-interactive.* Its fundamental characteristic is a high degree of interactivity between user and computer. We can consider current styles, including direct manipulation, to be essentially turn-taking ("ping-pong style") dialogues with a single input/output stream. Even where there are several devices, the input is treated conceptually as a single multiplexed stream, and interaction proceeds in half-duplex, alternating between user and computer. "Highly interactive" interfaces are characterized by continuous interaction between user and computer via several parallel, asynchronous channels or devices. They represent a change from half-duplex to full-duplex (i.e., allowing simultaneous user and computer actions) and from one serialized to several parallel input streams (e.g., meaningful eye input is obtained even while the user is operating a mouse or keyboard). Inputs are also often continuous, in contrast to the discrete, pre-tokenized input obtained from a keyboard and also typically of much higher bandwidth than traditional input devices. This type of input also often arrives in a very raw form, far from that necessary for use in a meaningful dialogue; it must be recognized or otherwise processed before being used in a dialogue.

An intermediate category on this axis can be defined as just "full-duplex," but not "highly interactive." This is a conventional turn-taking dialogue superimposed upon an environment in which other changes occur without user action. Examples include air traffic control, military command and control, and most graphics-based one-player videogames. In each, the base dialogue style is half-duplex, but additional changes in the displayed situation may occur concurrently with the dialogue, even when it is nominally the user’s turn.

New Interface Styles

The intersection of these two attributes, non-command and highly interactive, characterizes the next generation of user interface style. In it, commands are received implicitly, and continuous interaction is conducted in parallel over a variety of channels. Though the two attributes are somewhat correlated and often co-occur in new interfaces, considering each in isolation is helpful to provide a more precise understanding of an otherwise fuzzy notion of an emerging new interaction style. For example, practical uses of implicit inputs usually require that the computer monitor the user on a variety of continuous channels simultaneously, hence they are also most often "highly-interactive" by this definition. Some specific loci within this categorization are described in Figure 11.

Implications for User Interface Software

The characteristics of this new style of interface militate for new software architectures and approaches in user interface management systems and new techniques in user interface description languages to support them. This style will levy several new requirements on the next generation of underlying software, hardware, and UIMSs [9]:

- The new style requires much larger input and output bandwidth than that seen in the event queues of more conventional interfaces.
- With many new input devices, translation of the raw device inputs into dialogue tokens is a nontrivial problem; in cases such as speech or handwriting, it is an unsolved problem. The translation of raw keyboard input into meaningful tokens is so trivial (perhaps just consisting of translating shift key combinations into single characters) that this aspect of processing user input has been beneath consideration. Input data reduction in the future may occupy a large fraction of the available computer processing power. In addition, the process is not entirely open-loop, as recog-

nition systems often use context to improve their performance.

- It will also be necessary for the UIDL (user interface description language) to handle probabilistic tokens, where the recognizer passes a probability vector of possible tokens to the syntactic component of the UIMS, rather than a single token.

- A highly-interactive user interface must often respond continuously to user's input, rather than waiting for discrete tokens. Since any continuous input on a digital computer is ultimately quantized and processed discretely, the important distinction here is in the *user's* view of the dialogue, and hence the need is for explicitly handling continuous dialogues in the UIDL, not in the actual processing.

- Current UIMS technology typically handles multiple input devices by serializing all their inputs into one common stream. This is well suited to conventional dialogue styles, but is less appropriate for styles where the inputs are logically parallel (that is, where the user thinks of what he or she is doing as two simultaneous actions). Such parallel dialogues could of course be programmed within the old model, but it would be preferable to be able to describe and program them in the UIDL in terms of logically concurrent inputs, rather than a single serial token stream.

- The parallel input streams are not entirely independent either. Simultaneous or near-simultaneous inputs from different devices may need to be interpreted with reference to one another. The user might say, "delete that object" while pointing or looking at something. The spoken "that" must be interpreted in the context of the gesture the user was making at the time he or she said "that," not at the time the speech was finally processed. Inputs from each of the concurrent channels must thus be time-stamped and then later interpreted with respect to inputs on the other channels that occurred at the same time.

- The highly-interactive interface style requires excellent real-time response to the user's actions to be convincing (e.g., updating of a head-mounted display). This will require tighter and more explicit management of time and computing cycles than is common in current UIMSs. Deadline-based scheduling is important here, as are algorithms that can continuously adapt their "quality" to meet an impending deadline, making optimal use of whatever time is available to improve the output, but reliably completing within deadline.
- In general, time (as opposed to sequence) has not been explicitly called out in user interface specification or implementation. Syntax in existing UIDLs is considered to be the sequence of inputs and outputs, irrespective of their actual times. Notations and concepts for handling specific time in a UIDL will be needed.

Beyond Window Systems

Window systems provide a layer of support just below that of the UIMS. In moving to newer styles of interaction, not necessarily involving window or desktop displays, it is useful to think about what a "window system" could mean in a new, non window-based interface. To do so, we must first seek a more fundamental rationale for a window system. The essence of a window system is not that it carves up the available phosphor area, but, rather that it takes a set of user interface-related services needed by many applications, pulls them out of the individual applications, and provides them in a common, generic way, thus amortizing their cost. This includes services needed individually by the application as well as those services needed to allow applications to coexist. What would be the appropriate analogue for the new interface style? One example might be a three-dimensional volume manager, rather than a window manager [6]. For virtual reality systems, a layer incorporating modelling of physical motion is likely to be useful, since most applications will need to share such services. In fact, with an appropriate such layer, many of the virtual reality "applications" developed to date, which have been largely demonstrational, would be constructed simply from their UIDL syntax descriptions, without "application" code.

XII. CONCLUSIONS

Following Brooks' taxonomy [3], we present ''observations,'' rather than more formal ''findings'' of our experiences with eye movement-based interaction:

- 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. Obtainable accuracy is more similar to a traditional touch screen than a mouse, and the range can barely cover a single CRT display. The equipment, while non-intrusive and non-contacting, is difficult to ignore. Nevertheless, it is perhaps amazing that eye movement-based interaction can be done at all; when the system is working well, it can give the powerful impression of responding to its user's intentions rather than his or her 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. For example, we tried and rejected long gazes because they are not natural eye movements, preferring to use gazes only as long as natural fixations. 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.
- Our research concentrated on interaction techniques that can naturally and conveniently incorporate eye movements into a user-computer dialogue, rather than on the underlying eye tracker technology itself. 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.

The next step in this research is to perform more controlled observations on the new techniques. Our first experiment will compare object selection by dwell time with conventional selection by mouse pick. The extraneous details of the ship display are removed for this purpose, and a simple abstract display of circular targets is used, as shown in Figure 10. In the experiment, one of the targets will be designated, and the subject's task is to find it and select it, either by eye with dwell time or mouse. Response time for the two methods will be compared. (Initial pilot runs of this procedure suggest a 30 per cent decrease in time for the eye over the mouse, although the eye trials show more variability.)

Finally, we can view eye movement-based interaction as an instance of an emerging new style of user-computer interaction. This style exhibits changes from explicit to implicit commands and from turn-taking, single-stream dialogues to simultaneous, parallel interactions. Such interfaces will levy new requirements on user interface software and on the languages used to describe user-computer dialogues to handle and describe complex and substantial input/output processing, simultaneous parallel inputs, continuous inputs and outputs, imprecise inputs, and real-time constraints.

XIII. 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 and Rex Hartson and Chris Schmandt for encouraging me to present these results. This work was sponsored by the Office of Naval Research.

XIV. REFERENCES

1. R.A. Bolt, ‘‘Gaze-Orchestrated Dynamic Windows,’’ *Computer Graphics* **15**(3) pp. 109-119 (August 1981).

2. R.A. Bolt, ‘‘Eyes at the Interface,’’ *Proc. ACM Human Factors in Computer Systems Conference* pp. 360-362 (1982).

3. F.P. Brooks, ‘‘Grasping Reality Through Illusion– Interactive Graphics Serving Science,’’ *Proc. ACM CHI’88 Human Factors in Computing Systems Conference* pp. 1-11, Addison-Wesley/ACM Press (1988).

4. S.K. Card, W.K. English, and B.J. Burr, ‘‘Evaluation of Mouse, Rate-controlled Isometric Joystick, Step Keys, and Text Keys for Text Selection on a CRT,’’ *Ergonomics* **21**(8) pp. 601-613 (1978).

5. H.D. Crane and C.M. Steele, ‘‘Generation-V Dual-Purkinje-image Eyetracker,’’ *Applied Optics* **24**(4) pp. 527-537 (1985).

6. S.K. Feiner and C.M. Beshers, ‘‘Worlds within Worlds: Metaphors for Exploring n-Dimensional Virtual Worlds,’’ *Proc. ACM UIST’90 Symposium on User Interface Software and Technology* pp. 76-83, Addison-Wesley/ACM Press, Snowbird, Utah (1990).

7. J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley, Reading, Mass. (1990).

8. F.A. Glenn and others, ‘‘Eye-voice-controlled Interface,’’ *Proc. 30th Annual Meeting of the Human Factors Society* pp. 322-326, Santa Monica, Calif. (1986).

9. M. Green and R.J.K. Jacob, ‘‘Software Architectures and Metaphors for Non-WIMP User Interfaces,’’ *Computer Graphics* **25**(3) pp. 229-235 (July 1991).

10. R.N. Haber and M. Hershenson, *The Psychology of Visual Perception*, Holt, Rinehart and Winston, New York (1973).

11. T.E. Hutchinson, K.P. White, W.N. Martin, K.C. Reichert, and L.A. Frey, ‘‘Human-Computer Interaction Using Eye-Gaze Input,’’ *IEEE Transactions on Systems, Man, and Cybernetics* **19**(6) pp. 1527-1534 (1989).

12. R.J.K. Jacob, ‘‘A Specification Language for Direct Manipulation User Interfaces,’’ *ACM Transactions on Graphics* **5**(4) pp. 283-317 (1986). http://www.cs.tufts.edu/˜jacob/papers/tog.txt [ASCII]; http://www.cs.tufts.edu/˜jacob/papers/tog.pdf [PDF].

13. R.J.K. Jacob, ‘‘Human-computer Interaction,’’ pp. 383-388 in *Encyclopedia of Artificial Intelligence*, ed. S.C. Shapiro, John Wiley, New York (1987).

14. M.A. Just and P.A. Carpenter, ‘‘A Theory of Reading: From Eye Fixations to Comprehension,’’ *Psychological Review* **87**(4) pp. 329-354 (1980).

15. J.L. Levine, ‘‘An Eye-Controlled Computer,’’ Research Report RC-8857, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y. (1981).

16. J.L. Levine, ‘‘Performance of an Eyetracker for Office Use,’’ *Comput. Biol. Med.* **14**(1) pp. 77-89 (1984).

17. J. Merchant, R. Morrissette, and J.L. Porterfield, ‘‘Remote Measurement of Eye Direction Allowing Subject Motion Over One Cubic Foot of Space,’’ *IEEE Trans. on Biomedical Engineering* **BME-21**(4) pp. 309-317 (1974).

18. R.A. Monty and J.W. Senders, *Eye Movements and Psychological Processes*, Lawrence Erlbaum, Hillsdale, N.J. (1976).

19. J. Nielsen, ‘‘Trip Report: CHI’90,’’ *SIGCHI Bulletin* **22**(2) pp. 20-25 (1990).

20. R.M. Pritchard, ‘‘Stabilized Images on the Retina,’’ *Scientific American* **204** pp. 72-78 (June 1961).

21. E. Rich, ‘‘Users are Individuals: Individualizing User Models,’’ *International Journal of Man-Machine Studies* **18** pp. 199-214 (1983).

22. C. Schmandt, M.S. Ackerman, and D. Hindus, ‘‘Augmenting a Window System with Speech Input,’’ *IEEE Computer* **23**(8) pp. 50-56 (1990).

23. I. Starker and R.A. Bolt, ‘‘A Gaze-Responsive Self-Disclosing Display,’’ *Proc. ACM CHI’90 Human Factors in Computing Systems Conference* pp. 3-9, Addison-Wesley/ACM Press (1990).

24. H.M. Tong and R.A. Fisher, ‘‘Progress Report on an Eye-Slaved Area-of-Interest Visual Display,’’ Report No. AFHRL-TR-84-36, Air Force Human Resources Laboratory, Brooks Air Force Base, Texas (1984). Proceedings of IMAGE III Conference.

25. E.R. Tufte, ‘‘Visual Design of the User Interface,’’ IBM Corporation, Armonk, N.Y. (1989).

26. C. Ware and H.T. Mikaelian, ‘‘An Evaluation of an Eye Tracker as a Device for Computer Input,’’ *Proc. ACM CHI+GI’87 Human Factors in Computing Systems Conference* pp. 183-188 (1987).

27. L.R. Young and D. Sheena, ‘‘Survey of Eye Movement Recording Methods,’’ *Behavior Research Methods and Instrumentation* **7**(5) pp. 397-429 (1975).

Figure 1. Possible approaches to eye movement-based interaction.

	Unnatural response	**Natural (real-world) response**
Unnatural (learned) eye movement	Majority of work, esp. disabled	N/A
Natural eye movement	Jacob (this chapter)	Starker & Bolt, 1990

Figure 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. The display during this time was a Sun window system, with a mail-reading window occupying the left half of the screen, message headers at the top left of the screen, and bodies at the bottom left, and a shell window covering the bottom right quarter of the screen

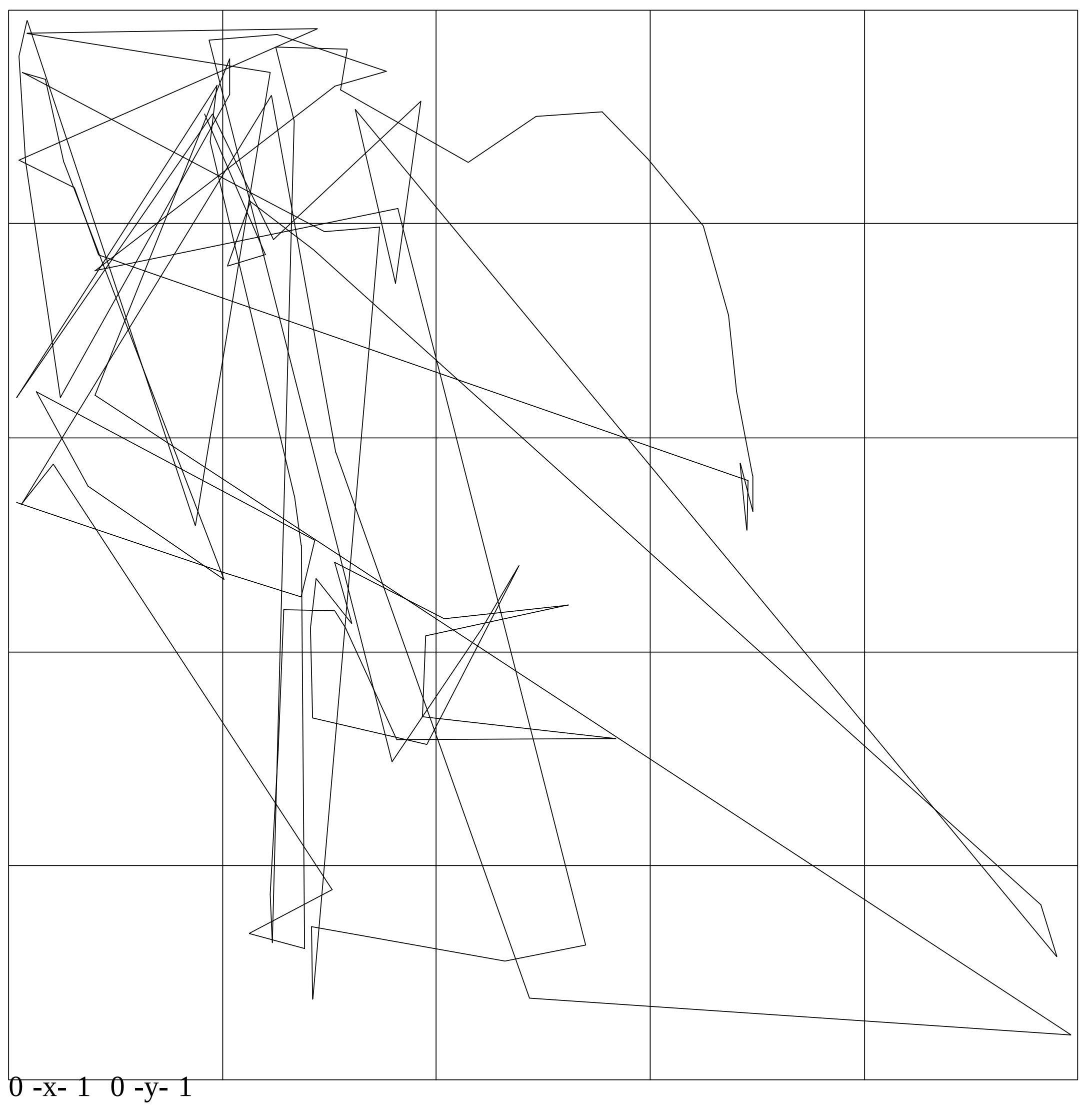


Figure 3. Illustration of components of a corneal reflection-plus-pupil eye tracker. The pupil camera and illuminator operate along the same optical axis, via a half-silvered mirror. The servo-controlled mirror is used to compensate for the user's head motions.

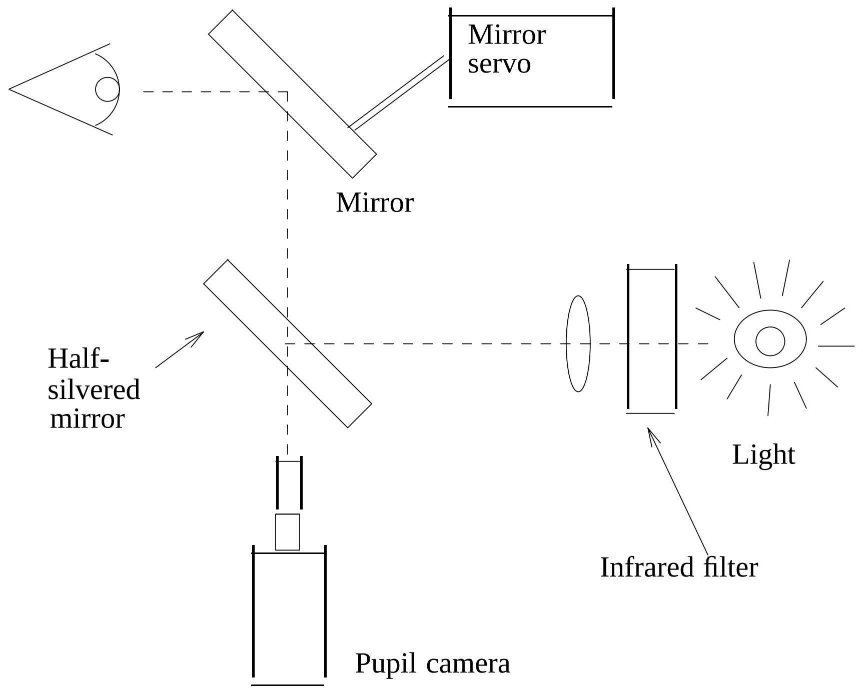


Figure 4. Illustration of erratic nature of raw data from the eye tracker. The plot shows one coordinate of eye position vs. time, over a somewhat worse-than-typical three second period.

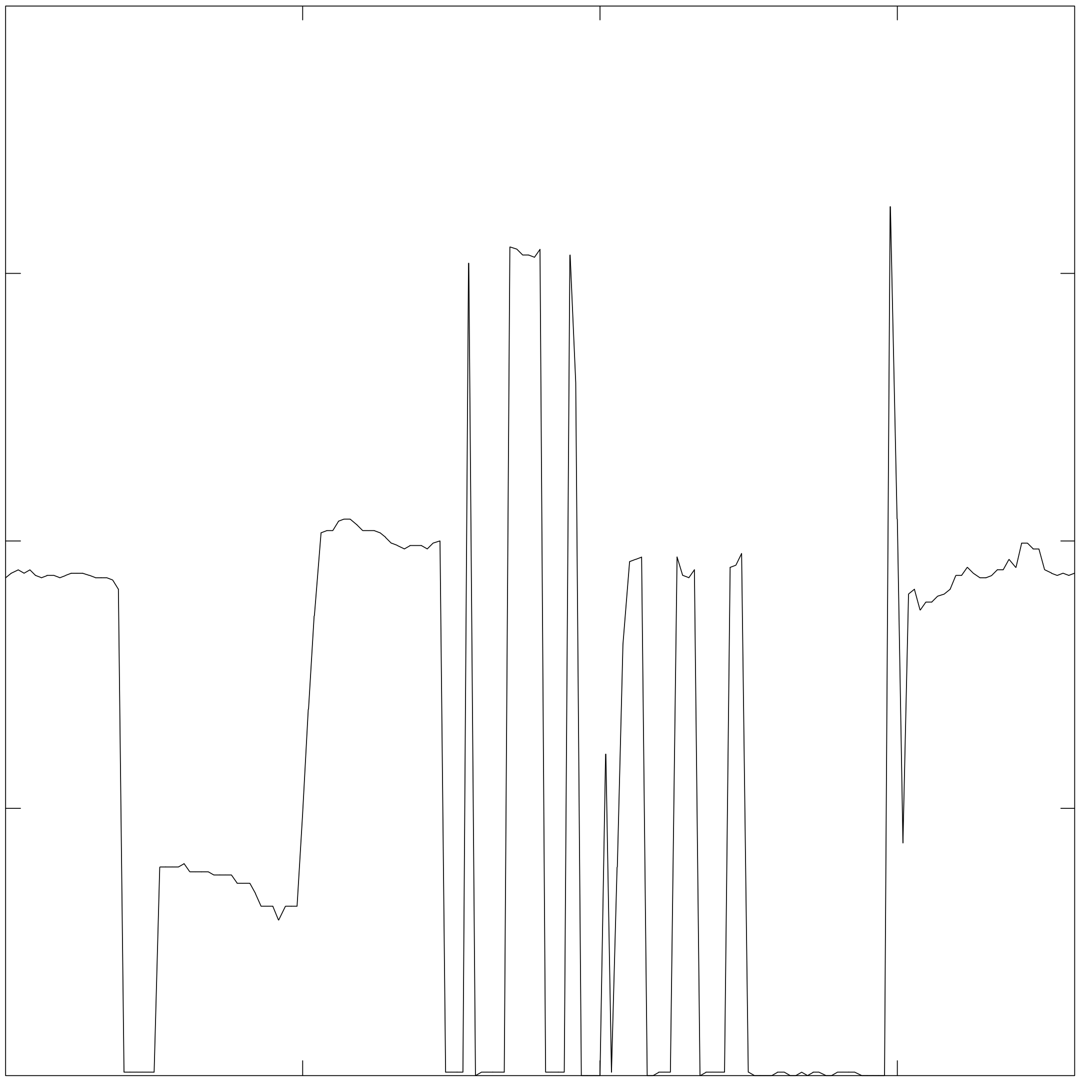


Figure 5. Result of applying the fixation recognition algorithm to the data of Figure 4. A horizontal line beginning and ending with an **o** marks each fixation at the time and coordinate position it would be reported.

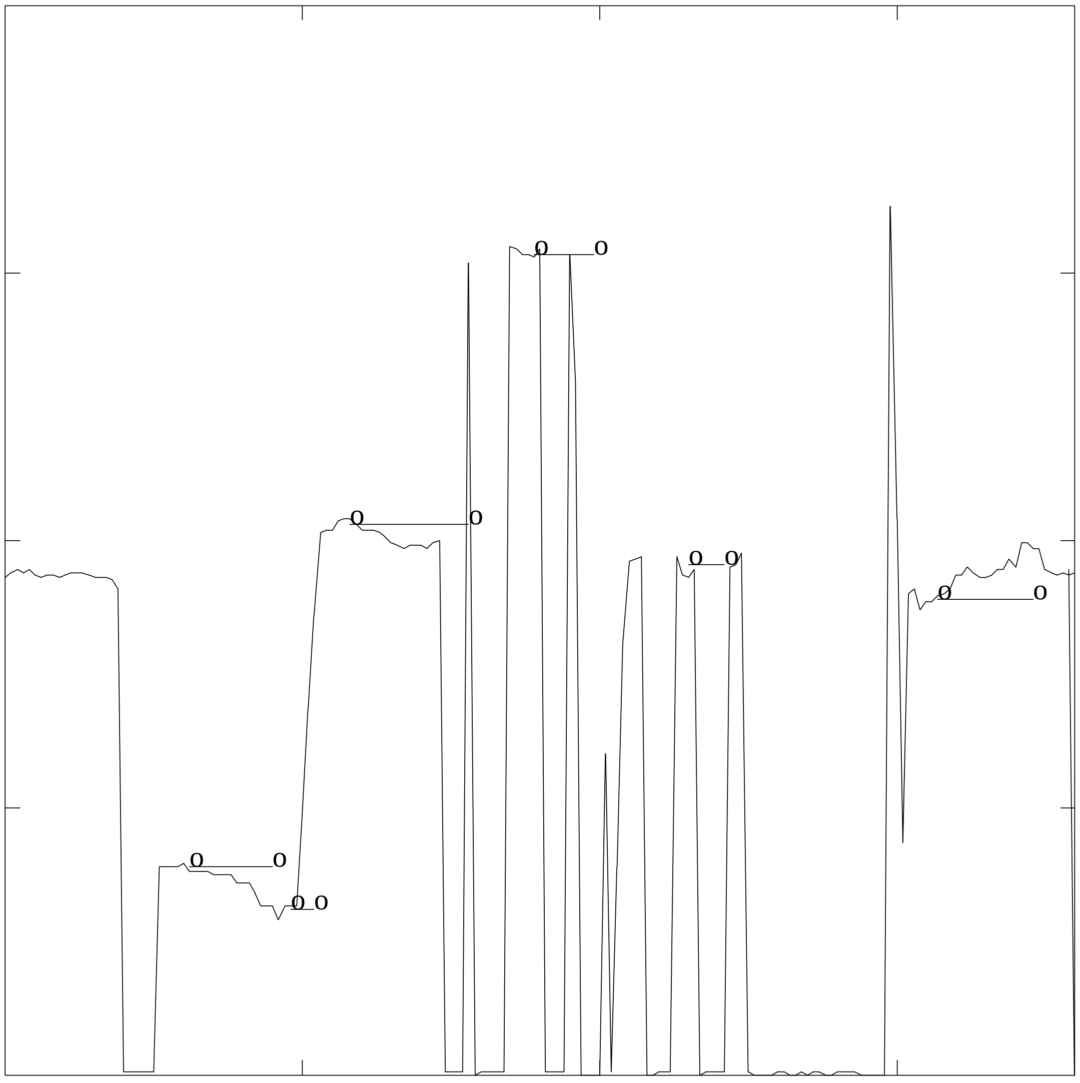
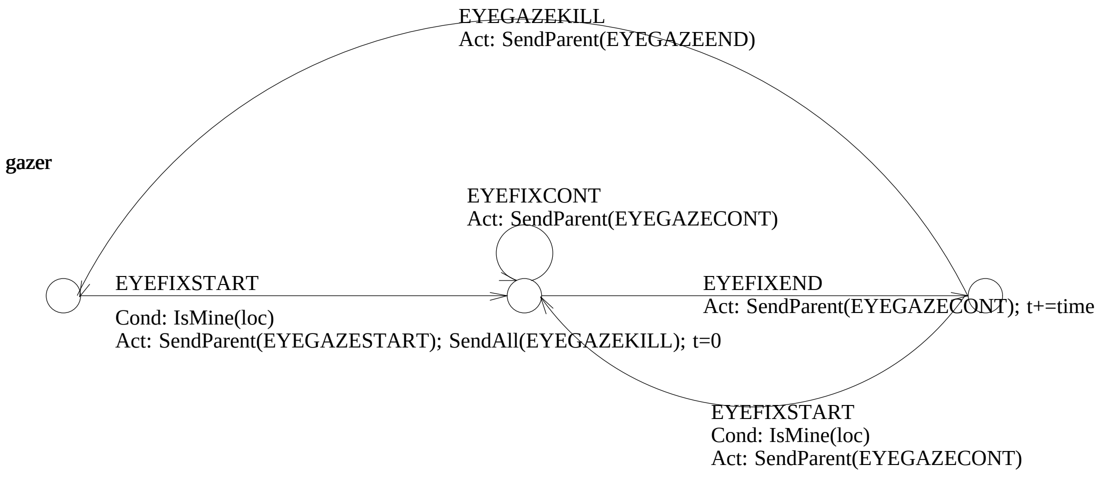


Figure 6. Display from eye tracker testbed, illustrating object selection technique. Whenever the user looks at a ship in the right window, the ship is selected and information about it is displayed in left window. The square eye icon at the right is used to show where the user's eye was pointing in 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.

(This figure is at the end of the file)

Figure 7. Syntax diagrams for the **Gazer** and **Ship** interaction objects.


EYEGAZEKILL
Act: SendParent(EYEGAZEEND)
gazer
EYEFIXCONT
Act: SendParent(EYEGAZECONT)
EYEFIXSTART
Cond: IsMine(loc)
Act: SendParent(EYEGAZESTART); SendAll(EYEGAZEKILL); t=0
EYEFIXEND
Act: SendParent(EYEGAZECONT); t+=time
EYEFIXSTART
Cond: IsMine(loc)
Act: SendParent(EYEGAZECONT)

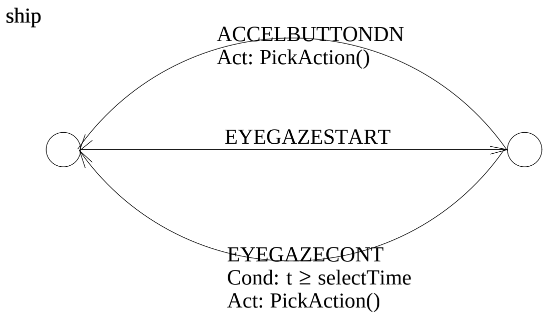


Figure 8. Another display from the testbed, showing the scrolling text and other windows.

(This figure is at the end of the file)

Figure 9. Testbed display showing eye-controlled pull-down menu.

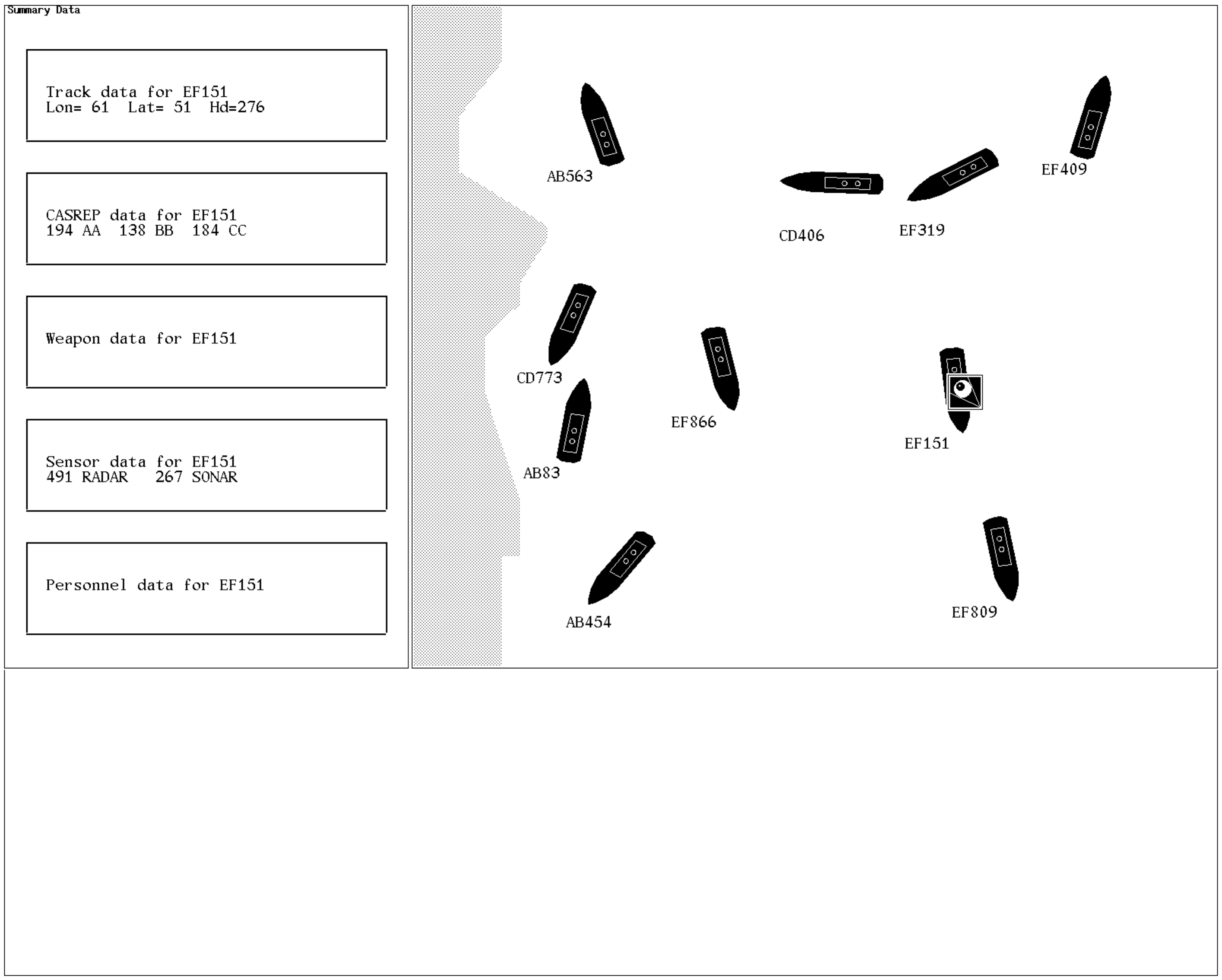
(This figure is at the end of the file)

Figure 10. 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).

(This figure is at the end of the file)

Figure 11. Characteristics of a new style of user-computer interaction.

		Command style	
Interactivity	Command-based (Explicit command)	Explicit command with re-interpretation	Non-command (Implicit command)
Half-duplex	Most current interfaces, from command language to direct manipulation. Despite differences, all respond to explicit commands from a single serialized input stream.		A somewhat artificial category; the commands are implicit, but they are received over a single channel. Examples include adaptive help or tutoring systems, single-channel musical accompaniment systems, and possibly eye movement interfaces that use no other input devices.
Full-duplex			
Highly-interactive	Familiar in everyday life, but less often seen in computer interfaces. Automobile or airplane controls and their computer-based simulators are good examples.		The next generation of user interface style. Current examples include virtual realities and multi-mode eye movement interfaces.



Summary Data

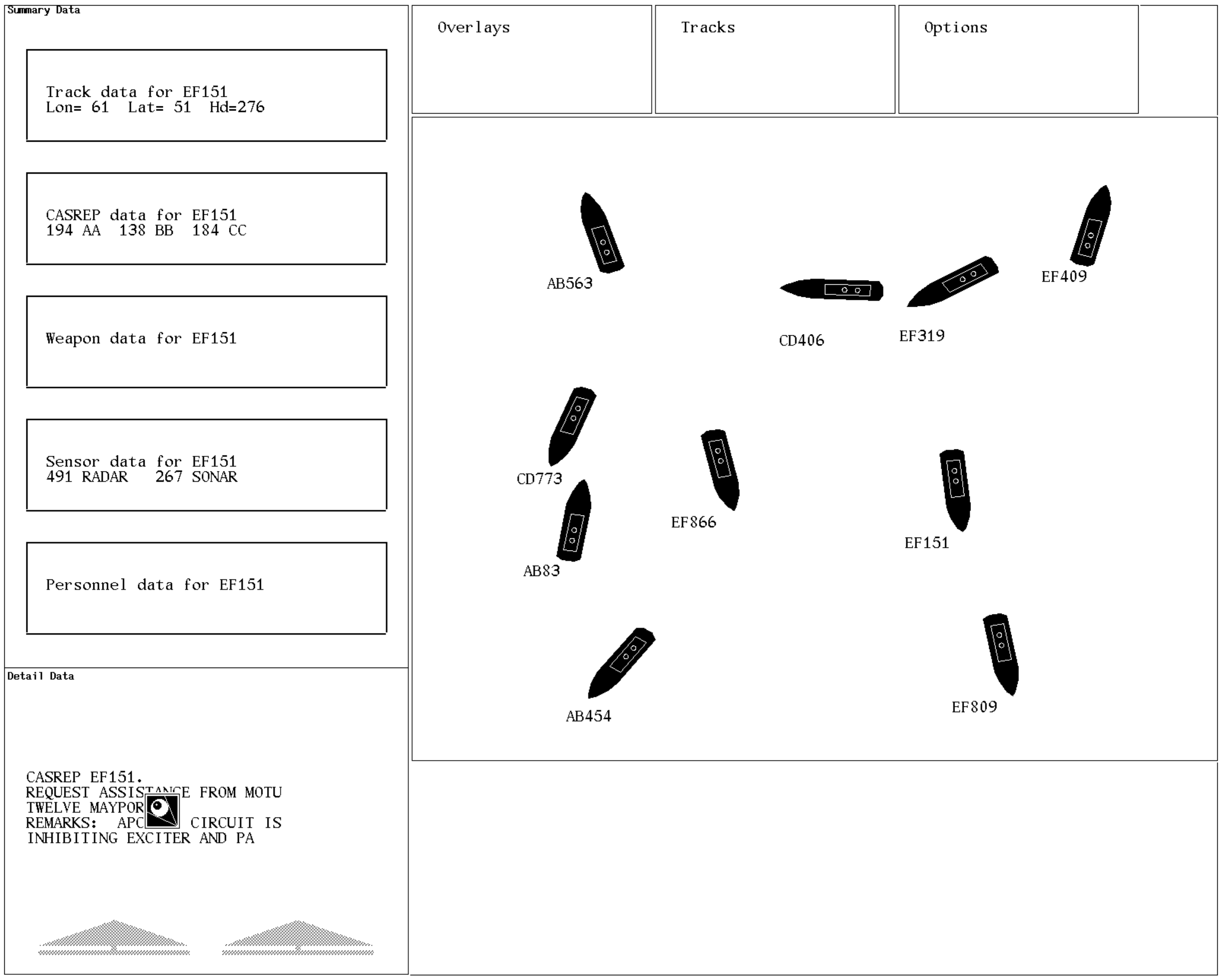
```
Track data for EF151
Lon= 61  Lat= 51  Hd=276
```

```
CASREP data for EF151
194 AA  138 BB  184 CC
```

```
Weapon data for EF151
```

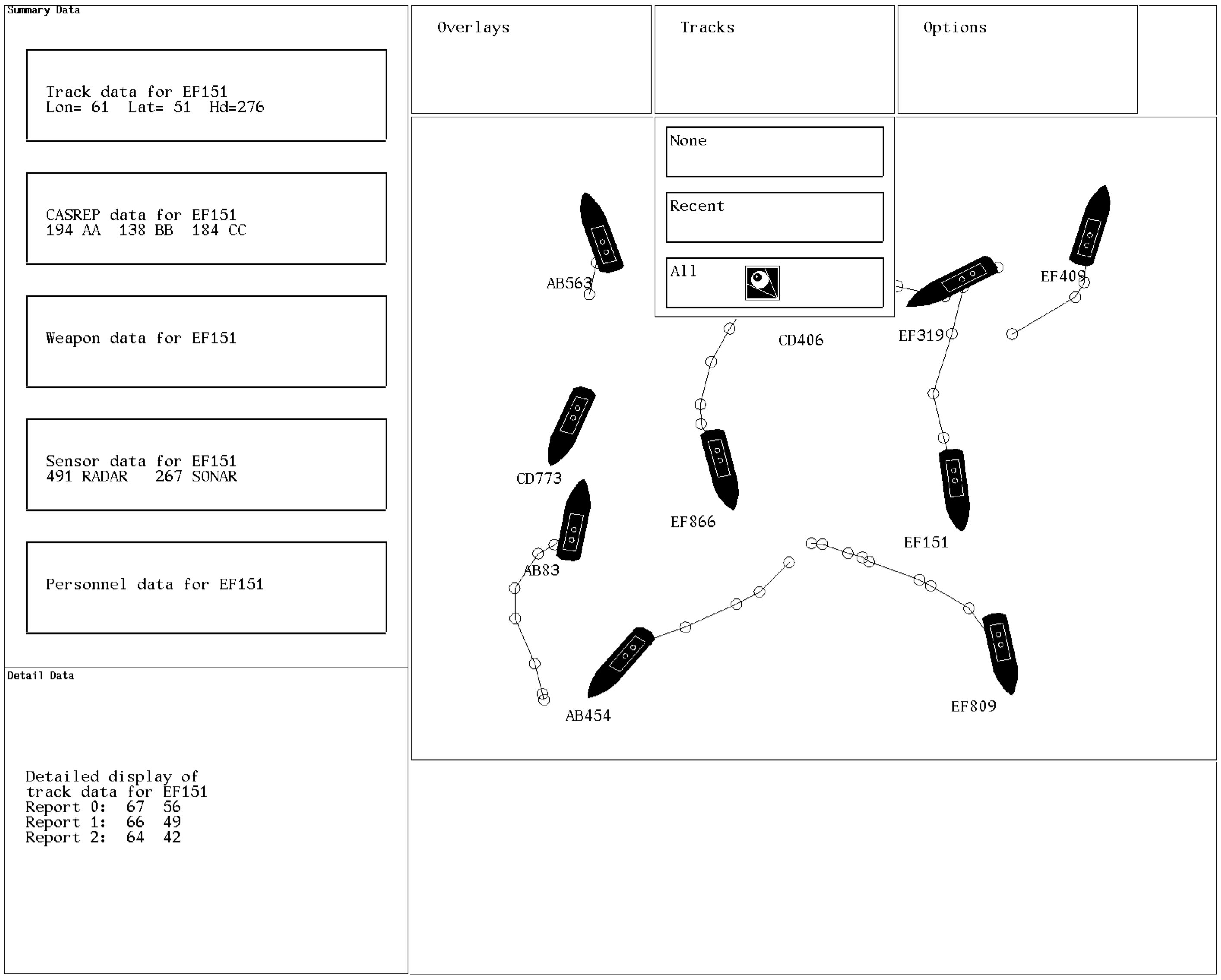
```
Sensor data for EF151
491 RADAR   267 SONAR
```

```
Personnel data for EF151
```



```
Track data for EF151
Lon= 61  Lat= 51  Hd=276

CASREP data for EF151
194 AA  138 BB  184 CC

Weapon data for EF151

Sensor data for EF151
491 RADAR   267 SONAR

Personnel data for EF151
```

```
CASREP EF151.
REQUEST ASSISTANCE FROM MOTU
TWELVE MAYPOR
REMARKS:  APC     CIRCUIT IS
INHIBITING EXCITER AND PA
```



Summary Data

```
Track data for EF151
Lon= 61  Lat= 51  Hd=276
```

```
CASREP data for EF151
194 AA  138 BB  184 CC
```

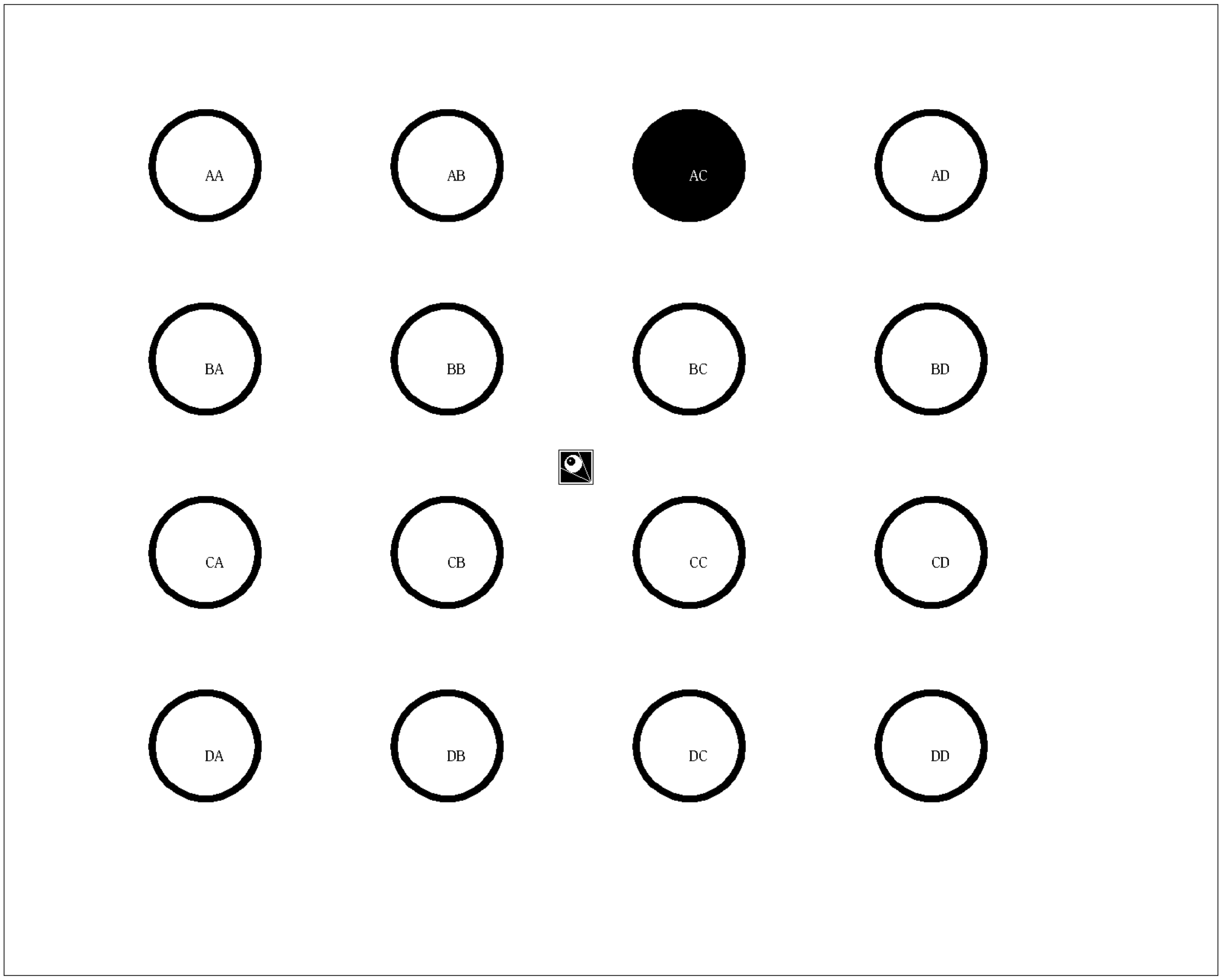
```
Weapon data for EF151
```

```
Sensor data for EF151
491 RADAR   267 SONAR
```

```
Personnel data for EF151
```

Detail Data

```
Detailed display of
track data for EF151
Report 0:  67  56
Report 1:  66  49
Report 2:  64  42
```


AA
AB
AC
AD
BA
BB
BC
BD
CA
CB
CC
CD
DA
DB
DC
DD