Eye Tracking in Advanced Interface Design

Robert J.K. Jacob

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

INTRODUCTION

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 [23]. To address it, we seek faster, more natural, and more convenient means for users and computers to exchange information. The user's side is constrained by the nature of human communication organs and abilities; the computer's is constrained only by input/output devices and interaction techniques that we can invent. Current technology has been stronger in the computer-to-user direction than user-to-computer, hence today's user-computer dialogues are rather one-sided, with the bandwidth from the computer to the user far greater than that from user to computer. Using eye movements as a user-to-computer communication medium can help redress this imbalance. This chapter describes the relevant characteristics of the human eye, eye tracking technology, how to design interaction techniques that incorporate eye movements into the user-computer dialogue in a convenient and natural way, and the relationship between eye movement interfaces and virtual environments.

Eye Movements and Virtual Environments

As with other areas of research and design in human-computer interaction, it is helpful to build on the equipment and skills humans have acquired through evolution and experience and search for ways to apply them to communicating with a computer. Direct manipulation interfaces have enjoyed great success largely because they draw on analogies to existing human skills (pointing, grabbing, moving objects in space), rather than trained behaviors. Similarly, we try to make use of natural eye movements in designing interaction techniques for the eye. Because eye movements are so different from conventional computer inputs, 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 trained eye movements to actuate the system. This requires careful attention to issues of human design, as will any successful work in virtual environments. The goal is for human-computer interaction to start with studies of the characteristics of human communication channels and skills and then develop devices, interaction techniques, and interfaces that communicate effectively to and from those channels. We thus begin with a study of the characteristics of natural eye movements and then attempt to recognize appropriate patterns in the raw data obtainable from the eye tracker, turn them into tokens with higher-level meaning, and design interaction techniques for them around the known characteristics of eye movements.

This approach to eye movement interfaces meshes particularly well with the field of virtual environments. The essence of virtual environment and other advanced interface approaches is to exploit the user's pre-existing abilities and expectations. Navigating through a conventional computer system requires a set of learned, unnatural commands, such as keywords to be typed in, or function keys to be pressed. Navigating through a virtual environment exploits the user's existing ‘‘navigational commands,’’ such as positioning his or her head and eyes, turning his or her body, or walking toward something of interest. By exploiting skills that the user already possesses, advanced interfaces hold out the promise of reducing the cognitive burden of interacting with a computer by making such interactions more like interacting with the rest of the world. The result is to increase the user-to-computer bandwidth of the interface and to make it more natural. An approach to eye movement interaction that relies upon natural eye movements as a source of user input extends this philosophy. Here, too, the goal is to exploit more of the user’s pre-existing abilities to perform interactions with the computer.

Moreover, eye movements and virtual environments both exemplify a new, non-command style of interaction. Some of the qualities that distinguish such interfaces 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 to, for example, movement of his or her eyes, head, or hands. Non-command-based interfaces will also have a significant impact on the design of future 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.

BACKGROUND

Physiology and Psychology of Eye Movements

If acuity were distributed uniformly across the visual field, it would be far more difficult to extract meaningful information from a person’s eye movement. Instead, receptors are distributed across the retina of the eye in a highly non-uniform way. The *fovea*, located near the center of the retina, is densely covered with receptors, and provides much higher acuity vision than the surrounding areas. The fovea covers approximately one degree field of view, that is, a one-degree angle with its vertex at the eye, extending outward into space. Outside the fovea, acuity ranges from 15 to 50 percent of that of the fovea. This peripheral vision is generally inadequate to see an object clearly; for example, to read a word of text generally requires that it be viewed foveally. It follows, conveniently for eye tracking purposes, that in order to see an object clearly, one must move the eyeball to make that object appear directly on the fovea. A person's eye position as measured by an eye tracker thus gives a positive indication of what he or she is viewing clearly at the moment. Though the density of foveal receptors is not uniform, it is sufficiently high over its approximately one degree area that one can obtain a clear view of an object anywhere in that area It follows, inconveniently for eye tracking purposes, that it is not possible to tell where within that approximately one-degree circle the person is looking. In fact, a person is able to pay attention to smaller area than the entire fovea and move this attention around within the fovea without making any eye movements. Thus, no matter how accurately an eye tracker can measure the eyeball position, we can infer the user's attention only to within the one-degree width of the fovea. Further, the relationship between what the person is viewing and what he or she is mentally processing is less straightforward, and, in particular, there can be brief time lags between the two. The eyeball is held in place by three pairs of opposing muscles, which provide motion in an up-down direction, left-right direction, and rotation around an axis from the fovea to the pupil. Interestingly, these six muscles provide little or no proprioceptive feedback to the brain; feedback comes in the form of changes to the image on the retina caused by the eye movements themselves.

The eye does not generally move smoothly over the visual field. Instead, it makes a series of sudden jumps, called *saccades,* along with other specialized movements [8, 25]. The saccade is used to orient the eyeball to cause the desired portion of the visual scene fall upon the fovea. It is a sudden, rapid motion with high acceleration and deceleration rates. It is ballistic, that is, once a saccade begins, it is not possible to change its destination or path. A saccade can cover from 1 to 40 degrees of visual angle, but most typically covers 15 to 20 degrees; it typically takes 30-120 milliseconds. The visual system is greatly suppressed (though not entirely shut off) during the saccade. Since the saccade is ballistic, its destination must be selected before movement begins; since the destination typically lies outside the fovea, it must be selected by lower acuity peripheral vision. If an object that might attract a saccade suddenly appears in peripheral vision, there is a 100-300 ms. delay before the saccade occurs. There is also a 100-200 ms. minimum refractory period after one saccade before it is possible to make another one.

More typically, a saccade is followed by a *fixation*, a period of relative stability during which an object can be viewed. Even during a fixation, the eye does not remain completely still, but engages in several types of small motions, generally within a one-degree radius. It drifts slowly, then is corrected by a tiny saccade-like jump (a microsaccade), which corrects the effect of the drift. In addition, it exhibits a high-frequency tremor, much like the noise in an imperfect servomechanism attempting to hold a fixed position. Fixations typically last between 200 and 600 ms., after which another saccade will occur.

Smooth motion of the eye occurs only in response to a moving object in the visual field. This smooth pursuit motion follows the moving object and is much slower than a saccade. Despite the introspective sensation that they eye is moving smoothly, this motion does not occur when viewing a static scene (or computer screen); it requires a moving stimulus.

Another, more specialized eye movement is called *nystagmus*. It occurs in response to motion of the head (particularly spinning) or viewing a moving, repetitive pattern. It is a sawtooth-like pattern of smooth motion that follows an object across the visual field, followed by a rapid motion in the opposite direction to select another object to follow, as the first one moves too far to keep in view. The motions described thus far are generally made by both eyes in tandem; the eyes also move differentially, making *vergence* movements to converge on an object. They point slightly toward each other when viewing a near object and more parallel for a distant object. Finally, *torsional rotation* occurs about an axis extending from the fovea to the pupil and is thus not detectable by a conventional eye tracker. It seems to depend on neck angle and other factors.

Despite introspective sensation to the contrary, one's eye is rarely stationary. It moves frequently as it fixates different portions of the visual field; even during a fixation it makes small motions; and it seldom remains in one fixation for long. In fact, if the effect of eye movements is negated by creating a fixed image on the retina (as the eye moves, the image moves with it) the image will appear to fade from view after a few seconds [19]. Normal eye movements prevent such fading from occurring outside the laboratory. Visual perception of even a static scene appears to require the artificial changes we induce by moving our eyes about the scene. The case of a user sitting in front of a desktop computer display is generally similar to viewing a static scene. We should expect to see steady (but somewhat jittery) fixations, connected by sudden, rapid saccades. We should not expect stationary periods longer than approximately 600 ms. We should not expect smooth motions unless a moving object appears in the display; and we are unlikely to observe nystagmus, vergence, or torsional rotation. For example, the history of 30 seconds of eye movements exhibited by a user working at a computer is shown in Figure 1. It contrasts sharply with a typical history of 30 seconds of mouse movements for most users.

Previous Work

There is a large and interesting body of research using eye tracking, but the majority of the work has concentrated on using eye movement data to provide a window on the operation of cognitive processes within the brain or of the eye movement control mechanism itself [12, 17]. Research of this type involves retrospective analysis of a subject's eye movements; the eye movements have no effect during the experiment itself. Our interest, in contrast, is in using eye movements to effect the user-computer dialogue in real time. There is a much smaller body of research and practice for this situation, much of it concentrated on helping disabled users, such as quadriplegics, who can move their eyes much more effectively than they could operate any other computer input device [9, 14, 15]. Because their ability to operate other devices is limited or nonexistent, the eye movement interface need perform only minimally well to provide a significant benefit to such users. Another similar situation occurs for a user whose hands are occupied, such as an airplane pilot. However, these cases hold the eye movement interface to a fairly low standard of performance, since the user has no competing alternative input media. An interface that would normally be rejected as too slow, awkward, or unnatural might still be effective in these cases. We seek instead to use eye tracking to provide an effective input method for a user whose hand might be resting on a mouse, who is fully able and available to operate the mouse, but who will find the eye movement based interface better, that is, faster, more convenient, or more natural.

There is a much smaller body of previous research in this area of eye movements, beginning with that of Richard Bolt, who has demonstrated innovative uses of eye movements in user-computer dialogues [1, 2, 21]. Floyd Glenn [6] demonstrated the use of eye movements for several tracking tasks involving moving targets. Ware and Mikaelian [24] reported an experiment in which simple target selection and cursor positioning operations were performed approximately twice as fast with an eye tracker than a mouse.

Perhaps the work most pertinent to ours is that of Starker and Bolt, which uses natural, real-time eye movements to affect the outcome of a user-computer dialogue. Their system ‘‘Analyzes the user’s patterns of eye movements and fixations in real-time to make inferences about what item or collection of items shown holds most relative interest for the user. Material thus identified is zoomed-in for a closer look and described in more detail via synthesized speech.’’ Their system displays a planet from the book, *The Little Prince.* In it, if the user glances around the scene in a general way, a general description of the planet is provided. If, however, the user looks at a pair of staircases, shifting glances between them, the system infers that the user is interested in staircases as a group rather than one specific one, and it provides an appropriate commentary. If the user glances principally at one staircase, then the commentary would describe that one specifically. ‘‘The generality or specificity of the system’s narrative is a function of the scope and focus of the user’s attention, whether wide and covering a group of items...or focused in upon some single thing, as inferred from the user’s pattern of eye fixations.’’ [21]

Another area where eye movements have been used in real-time interfaces is to create the illusion of a larger and higher resolution image than can actually be rendered. This has been demonstrated for flight simulator displays [22], and discussed for head-mounted displays. 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. If the eye tracker is good enough (and/or the coverage of the high-resolution inset is large enough), the user cannot detect the difference between this arrangement and the large, high-resolution display it emulates. Here, however, the eye movements are used essentially to simulate a better display device; the user’s eye movements do not alter the basic user-computer dialogue, as we wish to do.

TECHNICAL DESCRIPTION

Eye Tracking Technologies

A variety of technologies have been applied to the problem of eye tracking. Some are more suited to laboratory experiments or medical diagnosis than user-computer dialogues. Further details on this and the other eye tracking methods discussed here can be found in [25].

One of the least expensive and simplest eye tracking technologies is recording from skin electrodes, like those used for making ECG or EEG measurements. Because the retina is so electrically active compared to the rest of the eyeball, there is a measurable potential difference between it and the cornea. Electrodes are placed on the skin around the eye socket, and can measure changes in the orientation of this potential difference. However, this method is more useful for measuring relative eye movements (which require only AC skin electrode measurements) than absolute position (which requires more difficult DC measurements). It can cover a wider range of movement than other tracking technologies, but gives poor accuracy (particularly in absolute position). This method is principally useful for diagnosing neurological problems revealed in the nystagmus eye movements.

The most accurate, but least user-friendly technology uses a physical attachment to the front of the eye. A non-slipping contact lens is ground to fit precisely over the corneal bulge, and then slight suction is applied (mechanically or chemically) to hold the lens in place. Once the contact lens is attached, the eye tracking problem reduces to tracking something affixed to the lens, and a variety of means can be used. The lens may have a small mechanical lever, a magnetic coil, or a mirror, all of which can be tracked reliably. This method is obviously practical only for laboratory studies, as it is very awkward, uncomfortable, and interferes with blinking. It also covers a limited range of eye movements. However, it provides accurate data about the nature of human eye movements, which can be used to design effective interaction techniques and to make corrections to the data obtained from more practical, but less accurate tracking technologies.

Most practical eye tracking methods are based on a non-contacting camera that observes the eyeball plus image processing techniques to interpret the picture. The position of the eyeball can be identified by tracking one of its visible features. For example, the boundary between the sclera (white portion of the front of the eye) and iris (colored portion) is easy to find, but only a portion of it usually remains visible at one time. The outline of the pupil against the iris is a good choice if implemented properly. The eye is usually illuminated by barely-visible infrared light, so as not to be disturbing. Under infrared illumination, blue eyes appear dark and may make the black pupil difficult to identify. Shadows, dark eyelashes, and eyebrows may also interfere with identification of the black pupil. This can be alleviated with sophisticated image processing and pattern recognition techniques or, more simply, by illuminating the eye with a light that is coaxial with the camera. Under such illumination, the pupil appears as a bright disk, usually much brighter than surrounding features, regardless of eye color. For retrospective analysis, these approaches can be applied to movies or video tapes of the eye image. For real time analysis, the same approach is applied to a live video image of the eye.

As described, this approach measures the position of the front of the eyeball in space. For human-computer dialogues, we wish to measure *visual line of gaze*, a line radiating forward in space from the eye and indicating what the user is looking at. To illustrate the difference, suppose the eye tracker 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). One solution is to hold the head absolutely stationary, to be sure that any movement detected represents movement of the eye, rather than the head moving in space. This approach is used in laboratory experiments, but it requires a bite board, rather than a chinrest for good accuracy.

The more practical approach is to use simultaneous tracking of two features of the eye that move differentially with respect to one another as the line of gaze changes. This allows head movements (the two features move together) to be distinguished from eye movements (the two move with respect to one another). The head no longer need be rigidly fixed; it need only stay within camera range (which is quite small, due to the extreme telephoto lens required). The technique is to shine a collimated beam of infrared light at the front surface of the eyeball, producing a bright glint or corneal reflection, which moves less than the pupil as the eyeball rotates in its socket. The same infrared illumination can provide the corneal reflection and the bright pupil described earlier. The corneal reflection *and* outline of the pupil are then observed by the same video camera; image processing hardware or software analyzes the image to identify a large, bright circle (pupil) and a still brighter dot (corneal reflection) and compute the center of each. Then absolute visual line of gaze is computed from the relationship between these two points. The temporal resolution of this approach is generally 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 [3] tracks the corneal reflection plus the fourth Purkinje image (reflection from rear of lens of the eye); the latter is dim, so bright illumination of the eye is needed. The 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).

Current eye tracking technology is, thus, becoming suitable for ordinary users in settings outside the laboratory. Most of the methods discussed are more suitable for laboratory experiments, but the corneal reflection-plus-pupil outline approach is appropriate for normal computer users, since nothing contacts the subject and the device permits his or her head to remain unclamped. The eye tracker sits several feet away from the subject, and head motion is restricted only to the extent necessary to keep the pupil of the eye within view of the camera. A servomechanism pans and focuses the camera to follow the eye as the subject's head moves. Under ideal conditions, the subject can move within approximately one cubic foot of space without losing contact with the eye tracker. An alternative configuration, less convenient for normal office use, but more useful in virtual environments, is to mount a miniature version of the camera and light source directly on the user's head. That approach is described further in the next section.

This type of equipment is manufactured commercially in both the head-mounted and remote configurations; in our laboratory at NRL, we use an Applied Science Laboratories

(Waltham, Mass.) Model 4250R pupil to corneal reflection eye tracker [16, 25]. Figure 2 shows the components of this type of eye tracker. In our laboratory, the user sits at a conventional desk, with a 19" Sun computer display, mouse, and keyboard, in a standard office chair. 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 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 [10], multiplexes these tokens with other inputs (such as mouse and keyboard) and processes them to implement the user interfaces under study.

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. 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. Figure 3 shows the eye tracker in use in our interaction technique testbed.

Improvements in performance and cost of eye tracking equipment have come rather slowly in recent years. The accuracy of an eye tracker that is useful in a real-time interface is limited by the size of the fovea, since 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 eye movement research, but not for our purpose. 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. The accuracy of the best current eye trackers approaches one-degree useful limit imposed by the diameter of the fovea. However, *stability* and *repeatability* of the measurements leave much to be desired. In a research study it is acceptable if the eye tracker fails very briefly from time to time; it may require that an experimental trial be discarded, but the user need not be aware of the problem. In an interactive interface, though, as soon as it begins to fail, the user can no longer rely on the fact that the computer dialogue is influenced by where his or her eye is pointing and will thus soon be tempted to retreat permanently to whatever backup input modes are available. While eye trackers have dropped somewhat in price, their performance in this regard has not improved significantly. Performance does not appear to be constrained by fundamental limits, but simply by lack of effort in this area, due to its narrow market [25].

Most eye tracking techniques measure line of gaze, but not accommodation or focus, i.e., the specific point along the line of gaze at which the user has focused. For a user sitting in front of a flat display screen, this information is usually obvious. In addition, it is customary to track only one eye, since the two eyes generally point together,

Eye Trackers in a Virtual Environment

As an alternative to the remote camera and illuminator, a much smaller version of the camera and collimated light source can be mounted on the user's head, attached to a headband or helmet. The eye tracker then reports the angle of the user's eye with respect to his or her head. A separate magnetic tracker (such as a Polhemus tracker) measures head orientation, and the two data sources together can be used to determine the line of gaze in physical space. In virtual space, however, the head, eye tracker, and display itself all move together, so the head orientation information is not needed to determine line of gaze. The eye tracker range is the same as with the remote camera, but allowing head motion greatly expands overall range of the device, allowing the user to look anywhere around him or her, rather than on a single display screen. In addition, the relationship between camera and eye is more nearly constant with the head-mounted eye camera, offering the promise of better stability of eye tracking measurements. In our laboratory, we use a headband-mounted camera and light source as an alternative to the remote unit; the two produce similar video images of the eye and plug into the same video processing equipment. For many applications, the head-mounted camera assembly, while not heavy, is much more awkward to use than the remote configuration. However, in a virtual environment display, if the user is already wearing a head-mounted display device, the head-mounted eye tracker adds little extra weight or complexity. The eye tracker camera can obtain its view of the eye through a beam splitter, so it need not obscure any part of the user’s field of view. Depending on the display optics, some careful engineering may be required so as not to compromise very close viewing distances required by some wide-angle viewing optics; but this appears to be a soluble problem.

INTERFACE DESIGN ISSUES

Problems in Using Eye Movements in a Human-computer Dialogue

The other half of the application of eye movement input is to make wise and effective use of eye movements, ideally in a non-command-based style. Eye movements, like other passive, non-command inputs (e.g., gesture, conversational speech) are often non-intentional or not conscious, so they must be interpreted carefully to avoid annoying the user with unwanted responses to his actions. In eye movements, we call this the ‘‘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 cause an action to occur. At first it is helpful to be able simply to look at what you want and have it occur without further action; soon, though, it becomes like the Midas Touch. Everywhere you look, another command is activated; you cannot look anywhere without issuing a command. Eye movements are an example of how most of the non-command, passive inputs will require either careful interface design to avoid this problem or some form of ‘‘clutch’’ to engage and disengage the monitoring.

The simplest solution would be to substitute an eye tracker directly for a mouse. This is unworkable because of the way the eye moves as well as because of the instability of existing eye tracking equipment. Compared to mouse input, eye input has some advantages and disadvantages, which must all be considered in designing eye movement-based interaction techniques:

First, as Ware and Mikaelian [24] 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). 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 built-in 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.

During a single 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. Therefore, 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 requires 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.

In addition, 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 tracked as individual fixations. Following the same rationale, 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, if the user makes several fixations near the same screen object, connected by small saccades, the fixations are grouped together into a single ‘‘gaze,’’ following the approach of Just and Carpenter [12]. 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 is introduced into the output of the eye tracker whenever it 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.

Another simple interface design issue 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. Moreover, 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). 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. We often observe this phenomenon. Of course, 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).

We have divided the problem of processing eye movement data 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 other characteristics and imperfections of the eye tracking hardware, 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 more closely approximate 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. The second half of the process is to provide 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.

PROCESSING THE RAW EYE MOVEMENT DATA

Local Calibration

The first step in processing the raw data from the eye tracker is to introduce an additional calibration process. The eye tracker calibration procedure produces a mapping that is applied uniformly to the whole screen. No further calibration or adjustment should be necessary. In practice, we found small calibration errors appear in portions of the screen, rather than systematically across it. We introduce 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.

In our early experiments, this had the surprising 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. 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.

Fixation Recognition

The next step is to analyze the incoming data from the eye tracker and attempt to reconstruct the user's intentional fixations from the raw, jittery, imperfect data received. Such imperfections are 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.

A view of the type of data obtained from the eye tracker can be seen in Figure 4, which 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.

One solution would be 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.

Our algorithm is based on that used for analyzing previously-recorded files of raw eye movement data [4, 13] and on the known properties of fixations and saccades. It watches the input data for a sequence of 100 milliseconds during which the standard deviation of 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 therefore implies a delay of 100 ms. before reporting the start of a fixation. 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 cannot see it anyway.

If this algorithm is applied to the noisy data shown in Figure 4, we find 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

Thus far, the processing only translates eye tracker data into recognized fixations at specific screen locations without reference to what is displayed on the screen. The next processing 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.

USER INTERFACE SOFTWARE

The next step in the process of building a user interface is aided by our user interface management system (UIMS) [10], which lends itself to a multi-threaded direct manipulation interaction style typical of an eye movement-based interface. To use it, the output of the fixation recognition algorithm is turned into a stream of discrete *tokens* suitable for input to the UIMS. Tokens are reported 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, 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 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. All tokens, whether generated by eye, mouse, or keyboard, are then multiplexed into a single stream and presented as input to our UIMS.

To implement a user interface with this UIMS, the desired interface is specified as a collection of relatively simple individual dialogues, represented by separate *interaction objects.* The description of all of these interaction objects comprises the user interface description language (UIDL), which is executed by the UIMS. The interaction objects are connected by an executive that activates and suspends them with retained state, like coroutines. While the overall user interface may have multiple threads, each individual interaction object has only a single thread dialogue. A typical object might be a screen button, scroll bar, text field, or eye-selectable graphic object. Since 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 in the UIDL, and provides a standard method that the executive can call to offer it an input token and traverse its diagram.

The top level dialogue loop is provided by a standard executive. It operates by assembling the interaction objects and executing each of their state diagrams as a coroutine, 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.

As an example, each ship in Figure 6 responds to a gaze of a certain duration. Each of the ships is implemented as a separate interaction object (but all are of the same class, **Ship**). In addition, each interaction object such as **Ship** also has a lower-level **Gazer** interaction object associated with it to perform the translation of fixations into gazes described above. The **Gazer** accepts fixations on 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.

APPLICATIONS OF EYE TRACKING

Interaction Techniques

Interaction techniques are specific, yet not bound to a single application. An interaction technique is a way of using a physical input device to perform a generic task in a human-computer dialogue [5]. It represents an abstraction of some common class of interactive task, for example, choosing one of several objects shown on a display screen. Below, we describe the first eye movement-based interaction techniques that we have implemented and our observations from using them.

Object Selection

Object selection with a mouse is usually done by pointing at the object and then pressing a button to select one object from among several displayed on the screen. The selection might be one of several file icons on a desktop or, as shown in Figure 6, one of several ships on a map. However there is no natural counterpart of the mouse button press for the eye tracker. We reject using a blink because intentional blinking is not a natural eye movement. We examined two designs—using a button and using dwell time. With the button, the user looks at the desired object then presses a button to indicate that it is to be selected. In Figure 6, the user has looked at ship ‘‘EF151’’ and caused it to be selected (for attribute display, described below). With dwell time, the object is selected if the user continues to gaze at it for a sufficiently long time. 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; this allows the user to make a tradeoff between speed and keeping his or her hand free. In practice, however, we found the dwell time approach alone to be much more convenient, provided the dwell time could be made brief. A long dwell time would be good to ensure that the user will not make inadvertent selection simply by looking around on the display, but it attenuates the speed advantage of using eye movements for input and also reduces the responsiveness of the interface. To allow a reduced dwell time, we make a further distinction: If the result of selecting the wrong object can be undone trivially (that is, 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 we use a very short dwell time. 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. With a 150-250 ms. dwell time, this approach 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 into gazes. 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.

Continuous Attribute Display

This object selection interaction technique can be applied effectively to the retrieval of further details or attributes of one of the objects on a display. In Figure 6, we provide a separate area of the display for such attributes. The window on the right is a geographic display of ships, while the text window on the left displays attributes of one of the ships, as selected by the user's eye movement. 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. Because the text window is double-buffered, 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).

Moving an Object

If we separate the action of designating an object on the display to be moved from that of moving it, we might use the eye to select the object to be manipulated (moved on a map, in this case) and the mouse to move it. The eye selection is made as described above. 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.

As an alternative, the eye might be used 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.

Before testing, we expected that this 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

The bottom left of Figure 8 shows a window of text, which is not large enough to hold all the material to be displayed in it. A row of arrows is displayed 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

Since pop-up menus inherently assume a button, a pull-down menu seemed more appropriate for an eye movement-based interaction technique. 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.

A TAXONOMY OF EYE MOVEMENT-BASED INTERACTION

Direct manipulation and virtual environment interfaces both draw on analogies to existing human skills (pointing, moving objects, and navigating in physical space), rather than trained behaviors. 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. We draw two distinctions with respect to eye movement-based interaction, as shown in Figure 10: the nature of the user's eye movements and 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).

With regard to the eye movement axis, 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 10). 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.

The possible eye movement-based interaction techniques resulting from this categorization appear in Figure 10. 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 [21] 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 10, unnatural eye movement/natural response, is of doubtful utility.

NON COMMAND-BASED USER INTERFACES

Eye movement-based interaction and virtual environments are two of several areas of current research in human-computer interaction in which a new interface style seems to be emerging. This style 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 a *non-command-based* user interface: ‘‘This term may be a somewhat negative way of characterizing a new form of interaction, but the unifying concept does seem to be exactly the abandonment of the principle underlying all earlier interaction 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.'' [18]

Current examples of non-command-based interaction include eye movement interfaces, virtual environments, some music accompaniment systems, and agents. 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 [11, 20], we can view this distinction as *explicit* vs. *implicit* commands, thus non-command really means implicit commands.

Eye movement-based interaction provides an example of several of the characteristics—as well as the problems—of an emerging new user-computer interaction style that combines the non-command attribute with other somewhat correlated characteristics. This style is seen most dramatically in virtual environment 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 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 (see Figure 11).

One of these attributes is command style, 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.

The other attribute is interactivity, initially christened non-WIMP [7], 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). These interfaces attempt to avoid restricting the user's set of valid inputs, no matter what state the application is in. 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. Such interfaces thus require large amounts of input and output bandwidth, which leads to a need for devoting significant hardware resources to processing high bit rates of data as well as to data reduction and recognition algorithms needed to handle such voluminous, but imprecise inputs.

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 be initiated concurrently with the dialogue, even when it is nominally the user's turn.

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.

Within the other categories suggested by Figure 11, the *half-duplex/explicit command* category includes nearly all current interfaces, from command language to direct manipulation (excluding systems like command and control or air traffic control). Despite their differences, they all respond only to explicit commands from a single serialized input stream. The *half-duplex/implicit command* category is somewhat artificial, 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. The *highly-interactive/explicit command* category is familiar in everyday life, but less often seen in computer interfaces. Automobile or airplane controls are good examples; of course, computer-based simulators of those vehicles provide the same interface. Finally, the *highly-interactive/implicit command* category characterizes the next generation of user interface style. Current examples include virtual environments and multi-mode eye movement interfaces.

CONCLUSIONS

An eye tracker as an input device is far from ‘‘perfect,’’ in the sense that a mouse or keyboard is, and that is caused both by the limitations of current equipment and, more importantly, by the nature of human eye movements. Accuracy obtainable is more similar to a traditional touch screen than a mouse, and the range can barely cover a single CRT display. The equipment, while 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.

We saw that designing interaction techniques around a philosophy that emphasizes exploiting natural eye movements meshes well with the virtual environment interaction style, which is also based on natural navigational commands. Eye movement-based interaction and virtual environment interaction also share the non-command-based property, which will increasingly characterize advanced user-computer interfaces. And, in both areas, working from the natural characteristics of the human users to the interface designs leads to powerful improvements in the naturalness, convenience, and, ultimately, performance obtainable from advanced interface technologies.

Finally, we saw how eye movement-based interaction and virtual environments are instances of an emerging new style of user-computer interaction. This style is characterized by a change from explicit to implicit commands as well as a change from turn-taking, single-stream dialogues to simultaneous, parallel interactions.

ACKNOWLEDGMENTS

I want to thank my colleagues, Dan McFarlane, Preston Mullen, Diane Zimmerman, and particularly Linda Sibert, for their help with this research. I also thank an anonymous referee for some very helpful comments and observations on this chapter. This work was sponsored by the Office of Naval Research.

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

4. B.N. Flagg, ‘‘Children and Television: Effects of Stimulus Repetition on Eye Activity,’’ Thesis, Doctor of Education degree, Graduate School of Education, Harvard University (1977).

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

6. 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).

7. 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).

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

9. 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).

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

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

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

13. R.H. Lambert, R.A. Monty, and R.J. Hall, ‘‘High-speed Data Processing and Unobtrusive Monitoring of Eye Movements,’’ *Behavior Research Methods and Instrumentation* **6**(6) pp. 525-530 (1974).

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

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

16. 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).

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

18. J. Nielsen, ‘‘Noncommand User Interfaces,’’ *Comm. ACM* **36**(4) pp. 83-99 (April 1993).

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

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

21. 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).

22. 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.

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

24. 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).

25. 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. 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

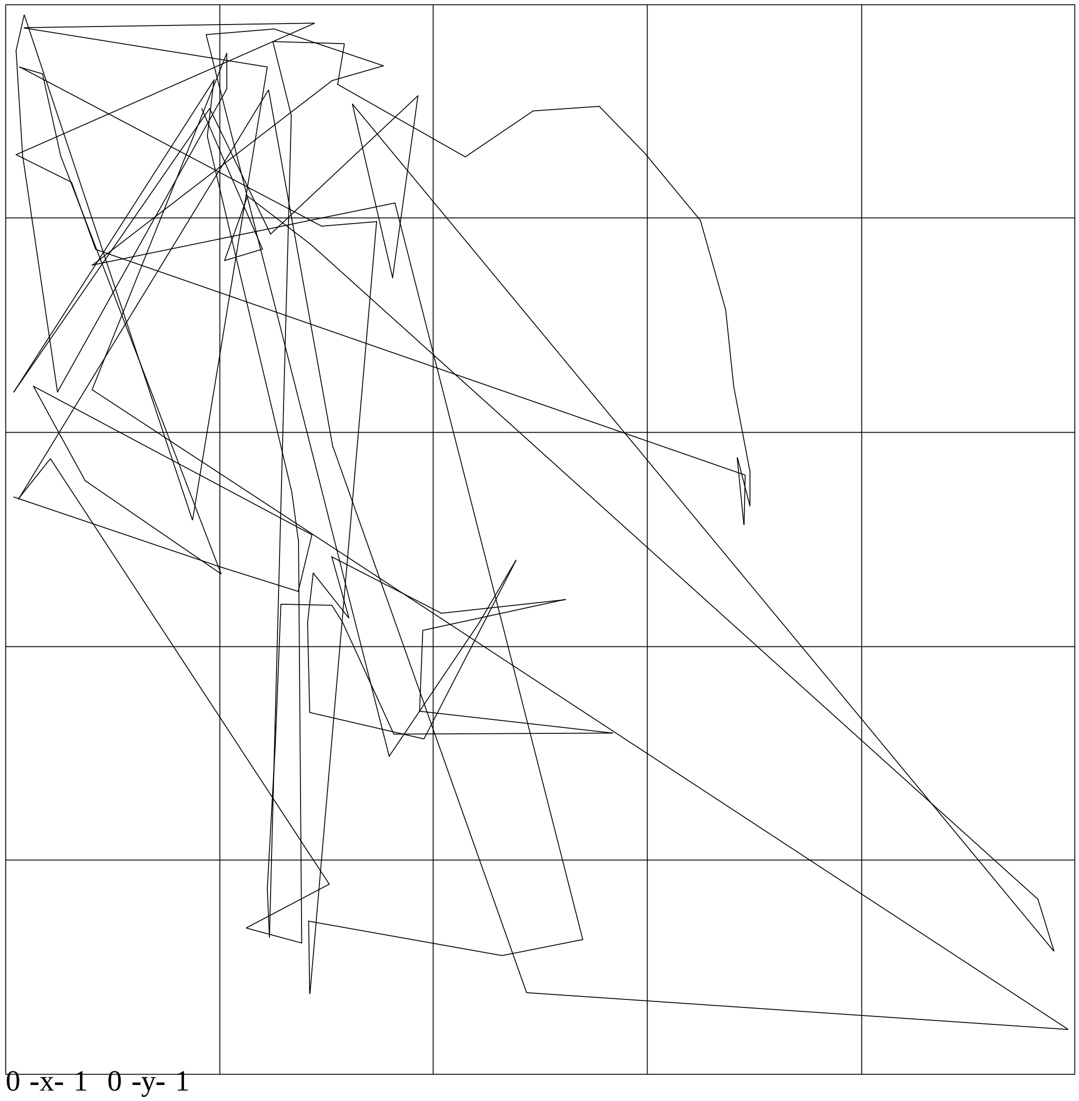

Figure 2. Illustration of components of a corneal reflection-plus-pupil eye tracker. The pupil camera and illuminator operate along the same optical axis, via a half-silvered mirror. The servo-controlled mirror is used to compensate for the user's head motions.

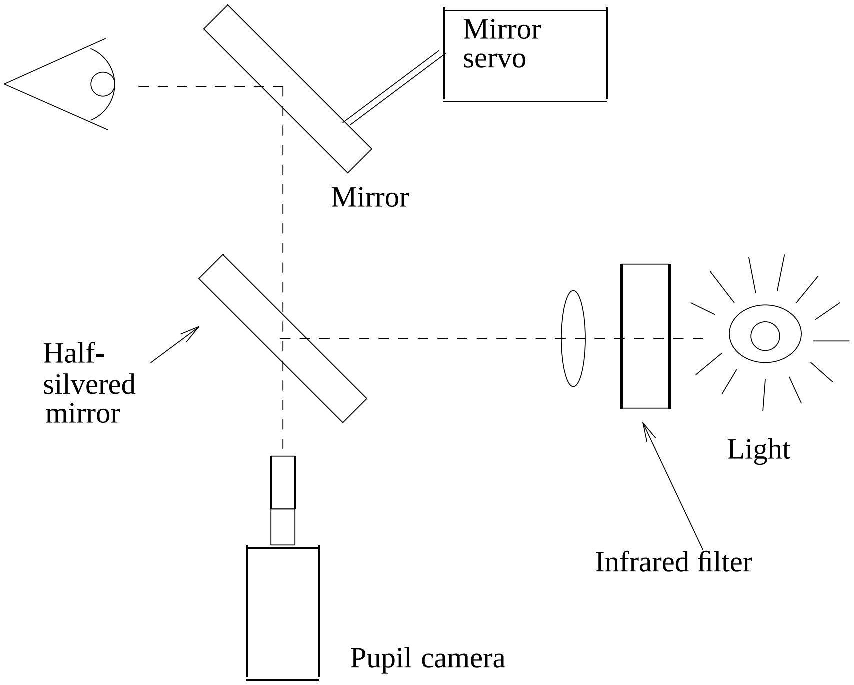

Figure 3. The eye tracker in use in the testbed. The eye tracker camera and light source are located at the lower left of this picture. The camera views the user’s eye through a mirror located just below the computer display screen. The eye tracker electronics are in the rack at the left of the picture, which also contains a monitor that shows the view of the user’s pupil as seen by the eye tracking camera.

(Insert photograph of eye tracker in use here)

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.

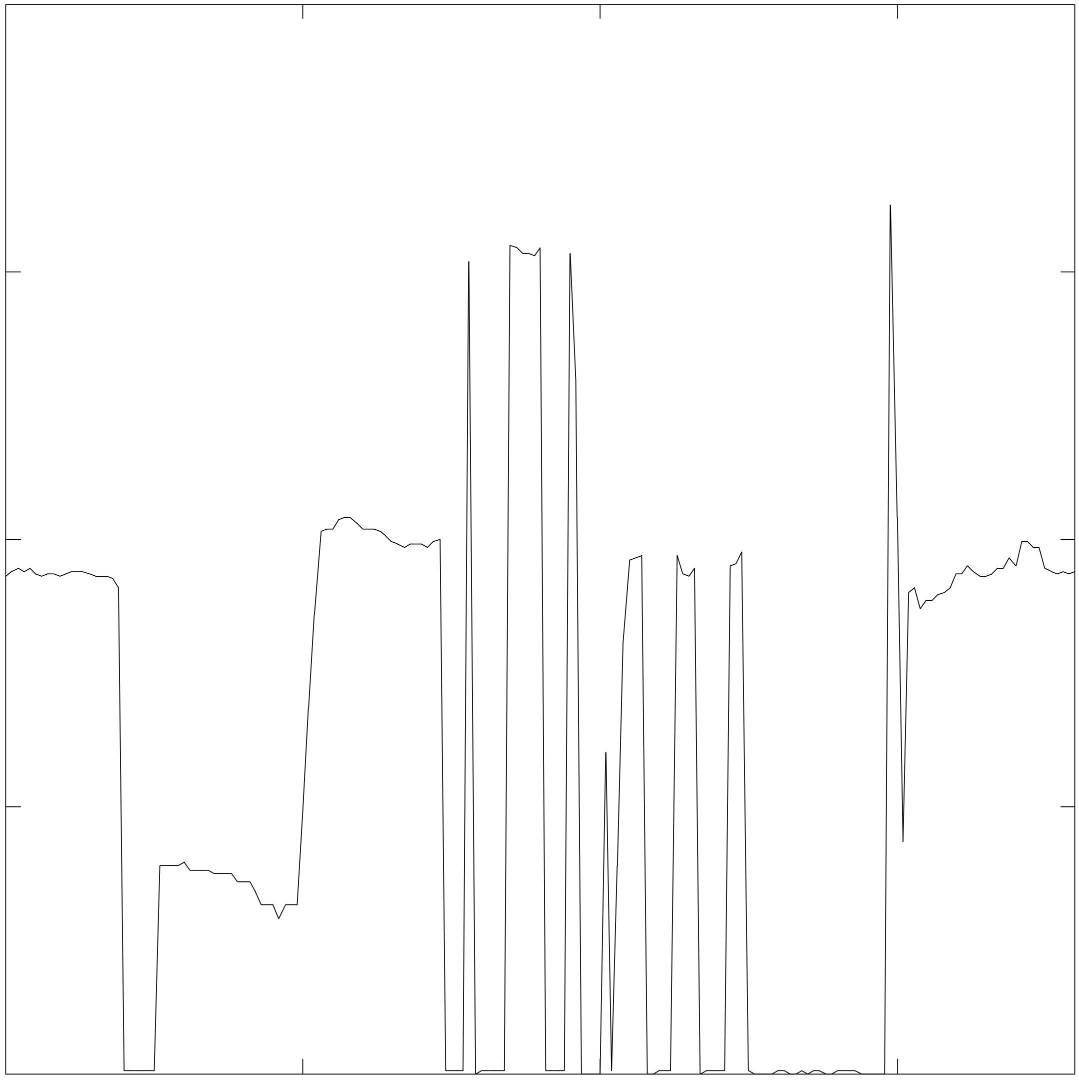

Figure 5. Result of applying the fixation recognition algorithm to the data of Figure 4. A horizontal line beginning and ending with an **o** marks each fixation at the time and coordinate position it would be reported.

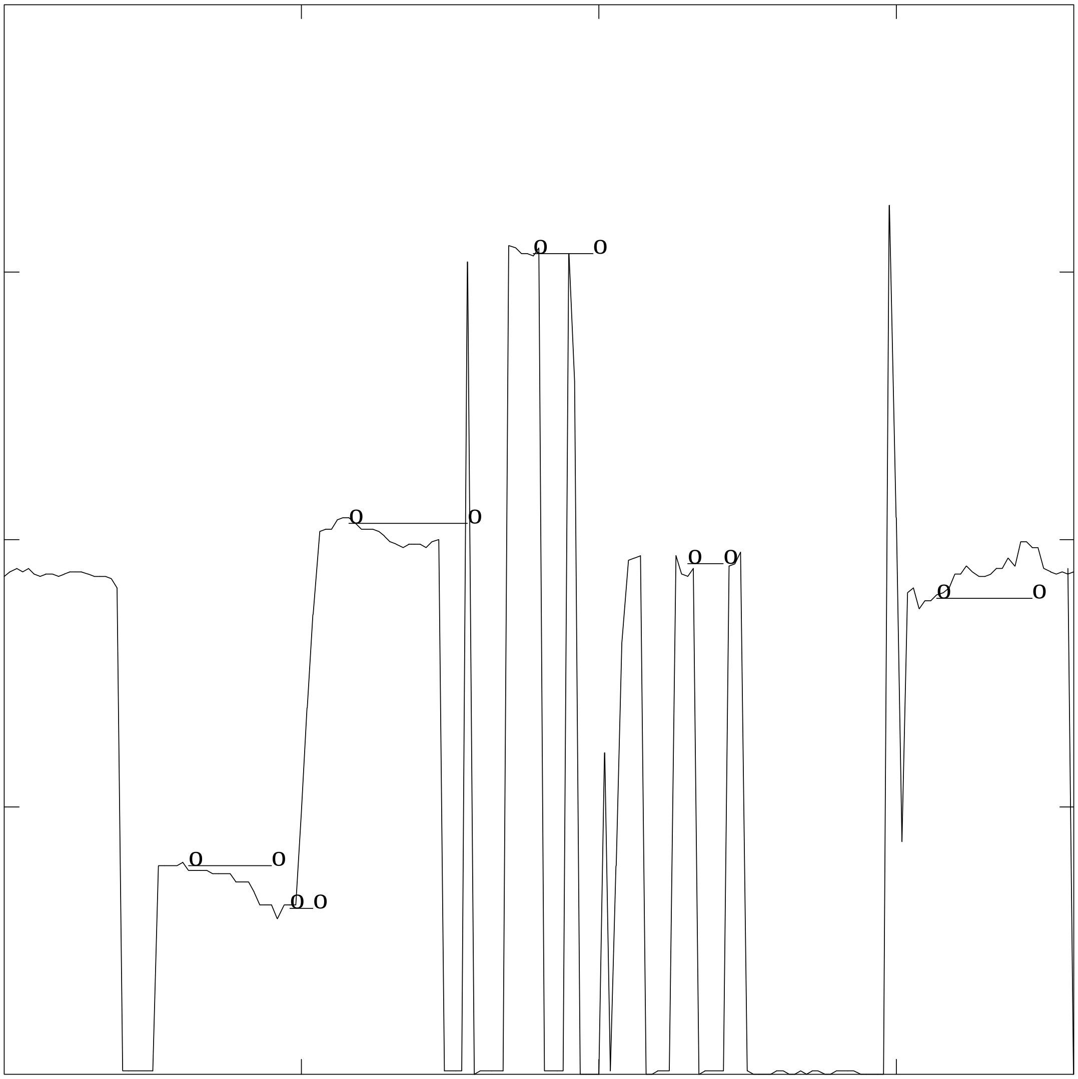

Figure 6. Display from eye tracker testbed, illustrating object selection technique. Whenever the user looks at a ship in the right window, the ship is selected and information about it is displayed in left window. The square eye icon at the right is used to show where the user's eye was pointing in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

Figure 7. Diagrams that specify the syntax of the **Gazer** and **Ship** interaction objects to the user interface management system.

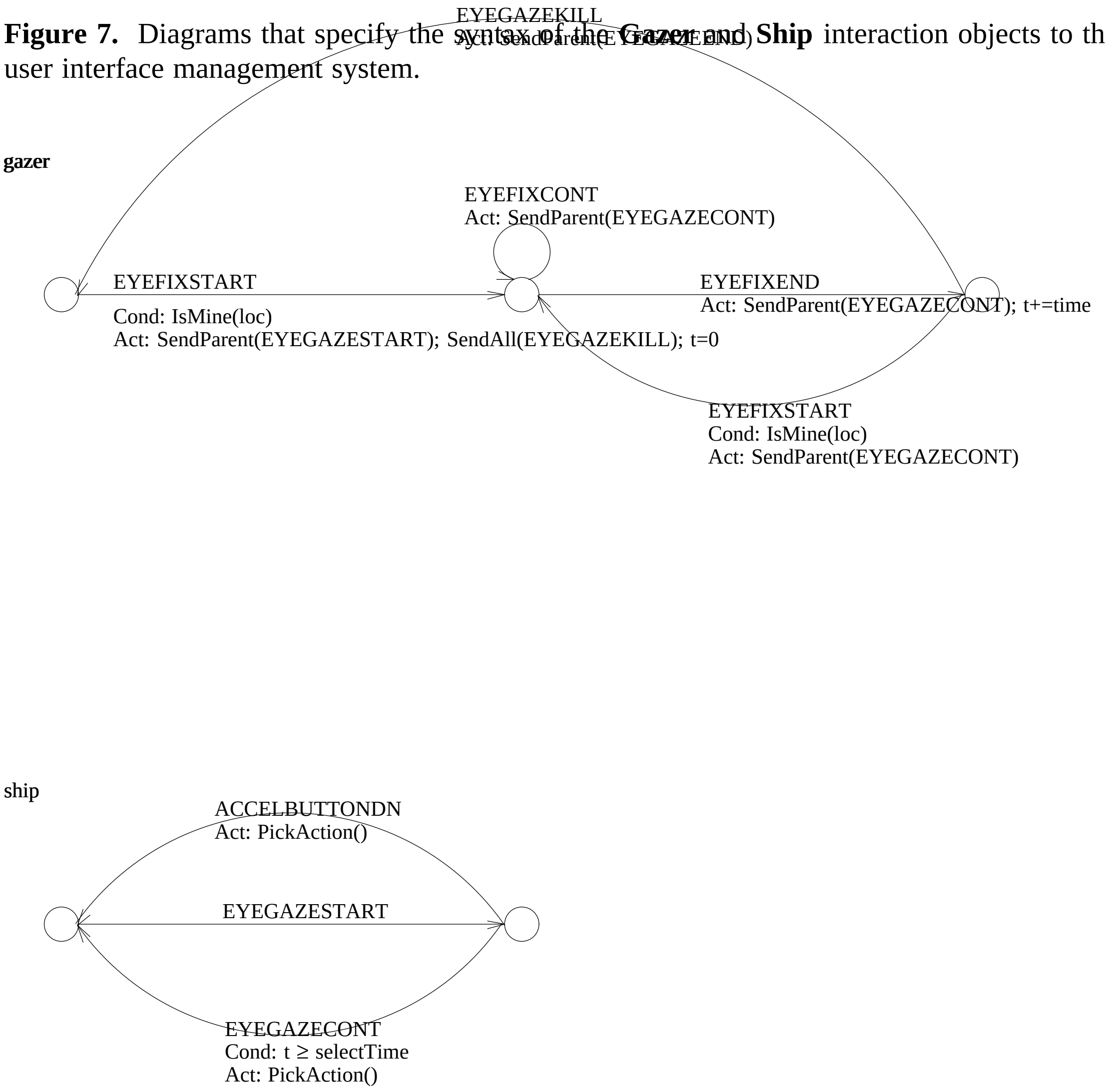

Figure 8. Another display from the testbed, showing the scrolling text and other windows.

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

Figure 10. A taxonomy of 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 11. A categorization of the components of a new style of user-computer interaction, and examples of some of them.

		Command style	
Interactivity	Command-based (Explicit command)	Explicit command with re-interpretation	Non-command (Implicit command)
Half-duplex	Most current interfaces	Command correction (DWIM)	Single-channel music accompaniment, eye-only interfaces
Full-duplex	Air traffic control, command and control		
Highly-interactive	Automobile controls, airplane controls simulators		Next generation: virtual environments, multi-mode eye interfaces