Real-time recording and classification of eye movements in an immersive virtual environment

Gabriel Diaz
Center for Perceptual Systems, University of Texas Austin, Austin, TX, USA

Joseph Cooper
Department of Computer Science, University of Texas Austin, Austin, TX, USA

Dmitry Kit
Department of Computer Science, University of Texas Austin, Austin, TX, USA

Mary Hayhoe
Center for Perceptual Systems, University of Texas Austin, Austin, TX, USA

Despite the growing popularity of virtual reality environments, few laboratories are equipped to investigate eye movements within these environments. This primer is intended to reduce the time and effort required to incorporate eye-tracking equipment into a virtual reality environment. We discuss issues related to the initial startup and provide algorithms necessary for basic analysis. Algorithms are provided for the calculation of gaze angle within a virtual world using a monocular eye-tracker in a three-dimensional environment. In addition, we provide algorithms for the calculation of the angular distance between the gaze and a relevant virtual object and for the identification of fixations, saccades, and pursuit eye movements. Finally, we provide tools that temporally synchronize gaze data and the visual stimulus and enable real-time assembly of a video-based record of the experiment using the Quicktime MOV format, available at http://sourceforge.net/p/utdvrlibraries/. This record contains the visual stimulus, the gaze cursor, and associated numerical data and can be used for data exportation, visual inspection, and validation of calculated gaze movements.

Introduction

Due to technological advances related to mobile eye-tracking, head-mounted virtual reality displays, and motion capture, the investigation of gaze behavior within virtual environments is increasingly feasible. However, only a handful of studies have adopted the analysis of gaze behavior using a head-mounted virtual display for the study of perception, action, and cognition in immersive virtual environments (Diaz, Cooper, Rothkopf, & Hayhoe, 2013; Duchowski, Medlin, Cournia, Gramopadhye, et al., 2002; Duchowski, Medlin, Cournia, Murphy, et al., 2002; Iorizzo, Riley, Hayhoe, & Huxlin, 2011). This is in part because the analysis of behavior within these environments presents a number of technical challenges. Therefore, we present this introductory guide for the use of eye-tracking technology in virtual environments with the aim of reducing the time and cost of startup. In addition, we present methods for both data capture and analysis of human gaze patterns when navigating a virtual environment.

To aid in data capture, we provide libraries that record the image stream presented to human subjects in real time along with a visual and text-based recording of data output from the eye-tracking software and virtual reality environment (http://sourceforge.net/p/utdvrlibraries/). Using this video record of the experimental process, experimenters are afforded immediate visual inspection of behavioral data, a numerical record for automated quantitative analysis, and a means for verifying that the data have been accurately interpreted and reconstructed following quantitative analysis.

We also provide users with several mathematical tools for transforming gaze data from pixel coordinates into forms commonly used in behavioral analysis. For example, we provide equations for calculating absolute gaze angles within a world-based reference frame. In this section, we also provide equations that allow one to compensate for the use of head-mounted displays...
(HMDs) with screens that have been angled away from fronto-parallel plane—a strategy sometimes used to increase the HMD field of view. In the next section, we present methods for calculating the angular distance from gaze to objects within the virtual environment. Finally, we present methodology for the identification of fixations, pursuit, and saccadic eye movements.

**Hardware**

As a minimum requirement for use of our algorithms and software, a laboratory will require three basic components: an HMD outfitted with a monocular eye-tracker, a motion-capture system, and a computer for delivery of the visual stimulus and data logging. Because these components are widely interchangeable and offered by a number of companies, we refrain from making specific hardware recommendations. However, we provide the details of our own system to serve as an example of the functional configuration.

Interactive virtual environments require a cyclical design in order to keep the stimuli presented to the subjects consistent with their movement and other behavior (Figure 1). Each cycle generates a frame of data that are recorded for later analysis. The visual stimulus is presented in an HMD worn by the subject that is outfitted with equipment necessary for tracking head movement and gaze within the virtual environment. In our work, we presented the visual stimulus on an Nvis SX111 HMD with a field of view that stretches 102° along the horizontal and 64° along the vertical. A 14 camera Phasespace X2 motion capture system running at 480 Hz tracked head and body movement within a capture volume with submillimeter accuracy.

The total latency before a physical movement was updated onscreen was approximately 50 ms. To produce this measure, an oscilloscope monitored voltage output from a photodiode, and from a constant voltage-weighted electronic device. This device was affixed with motion-capture markers and secured to a low-friction vertical track. The testing procedure involved dropping the device so that it collided with the ground plane. This collision would activate a switch on the bottom of the electronic device, causing an instantaneous step change in voltage output. Simultaneously, the Phasespace system tracked the motion of the device during its downward descent. At the time that the device would reach the bottom-most position on the track, the HMD screen would change from black to white, bringing about a global change in screen luminance inside the HMD. This change was detected by the photodiode and brought about a step change in voltage reflected on the oscilloscope. Ten repeated tests all yielded a constant latency of three visual frames (approximately 50 ms) between the step-change in voltage from the electronic device and the resulting on-screen change in luminance as detected by the photodiode.

An infrared, video-based eye-tracker from Arrington Research (Scottsdale, AR) tracks the eye within the HMD and is accompanied by Viewpoint software version 2.9.2.6. The eye-tracker refresh rate is restricted by the NTSC standard and functions at a frequency of approximately 60 frames/second when the image is de-interlaced. Although the process of de-interlacing is accompanied by some loss of resolution, it is typical of all modern eye-tracking suites to have adopted algorithms that are robust to this degradation.

Because the quality of eye-tracker calibration will directly influence the quality of data, we highly suggest that the quality of the track is inspected at regular intervals throughout an experiment. The frequency of recalibration should be tailored to the experimental demands and is dependent upon the amount of movement the subject undergoes, the rigor of movement, the weighting and mounting of the HMD, and the level of precision required during analysis.
Programs for stimulus presentation and data collection are run on a primary computer and controlled via a primary display. In addition, the images presented to the subject’s left and right eyes via the HMD are mirrored on two tabletop displays visible to the experimenter. To simulate movement through the virtual world, the subject’s head position must be tracked using a motion capture system. To ensure low-latency motion data, the motion capture data are typically collected and processed on a dedicated server, which may be polled by the experimental machine via ethernet prior to updating the visual stimulus. Our primary computer for stimulus generation and data collection was a PC with an Intel Q6600 processor (Intel, Santa Clara, CA) running Windows XP 32 bit (Microsoft, Bellevue, WA), with 4 GB of RAM and a two-way SLI configuration using two NVIDIA GTX 9800 video cards (NVIDIA, Santa Clara, CA).

The UT Digital Video Recorder

One of the first technical hurdles a user will face is ensuring that the recorded gaze data and recorded locations of the objects in the virtual world accurately align the asynchronous streams related to motion tracking, stimulus generation, and the monitoring of gaze data. Consider that the speed at which the visual stimulus is refreshed shares neither frequency nor phase with the rate at which gaze data is made available by the eye-tracking software, an issue that is compounded by the common strategy for eye-tracking software suites to employ the use of data buffers. Thus, when data are requested, one may receive outdated information, multiple information packets, or no data packets at all. If not handled properly, temporal misalignments will arise between the recorded gaze angles and the position of the head within the virtual world.

To aid in the process of data alignment, we provide the UT Digital Video Recorder (DVR) libraries for recording a digital video record of the on-screen image in real-time as an Open Source project for others’ use (http://sourceforge.net/p/utdvrlibraries/). The UT DVR libraries take advantage of the Quicktime 7 video format (Apple, Inc., Cupertino, CA) to allow for the temporal alignment of multiple video and text tracks within a single MOV format container. Since the visual scene is delivered to the HMD, the UT DVR program works in parallel to compress each frame on-screen image to a resolution of 640 × 480, to overlay the frame with a crosshair centered upon the pixel coordinate most recently returned by the eye-tracker data, and to append the frame into a movie track stored within the MOV container (Figure 2). In addition, a parallel text track is used to store quantitative information about the time at which the visual scene data were recorded, the location of the subject’s head and gaze, and any additional information provided by the user. In addition, a separate movie track is used to record the image of the subject’s eye seen through the eye-tracking camera, as well as visual feedback regarding the tracking quality that was provided by the eye-tracking software. This movie track is also paired with a text track to provide quantitative information regarding the gaze data, including the time at which the gaze data were collected. Finally, an additional text track records the time at which the eye data and scene data were written to the Quicktime movie container. To summarize, the final movie contains five tracks: the rendered video frame depicting the view presented to the left eye, with a crosshair at the subject’s gaze location; the video image of the subject’s eye; a text track containing user defined metadata; a text track containing the information about when the eye position information was retrieved; and a text track containing the time when all the tracks were combined.

The utility of this video-based record is multipurpose. The video provides a means for visual inspection of subject behavior that helps guide quantitative analysis of the data stream. Because this video represents minimally processed raw data, the video may later be used as a visual reference for comparison against data that have been processed during analysis. Once the user has begun this analysis, the text-based data stream can be extracted from the video using Matlab or Python, and subsequently used to reconstruct various features of the virtual environment, as seen from the viewer’s perspective. Mismatches between the reconstruction and video-based data may be attributed to errors of inference during data analysis.

Digital vide recorder: A technical description

The DVR program is written in both Python and C++, and combines the computational power of the GPU with a video encoder to provide an efficient method for writing experimental data to disk. It is because of this efficiency that the DVR code may be run in real time at rates below 60 Hz (based upon performance of the test machine), without interfering with the process of stimulus generation. The majority of the encoding process occurs immediately after OpenGL issues a post buffer swap call, at which time a 640 × 480 pixel frame buffer object is created. The graphics card renders a textured quad onto this framebuffer object using the contents of the most recently
drawn visual scene. Subsequently, the program paints a white crosshair around the subject’s point of gaze. To increase efficiency and reduce the load placed upon the CPU, the graphics card carries out resizing operations. A 60-frame buffer compresses multiple frames on the video card in parallel, ensuring rapid compression even if the frame rate exceeds 60 Hz for small periods of time. However, if the GPU cannot keep up with the rendering rates and this buffer is filled, the software blocks, temporarily reducing the overall frame rate.

Libraries have been tested on our standard machine (reported in Hardware), and on an Intel Core i7 920 running at 2.67 GHz with 6 GB of onboard memory, running Windows 7 with service pack 1, with a pair of NVidia GeForce 9800 GT graphics cards. The libraries have also been tested on the same machine using a pair of NVidia GeForce GTX 550 Ti graphics cards.

**UT DVR system requirements**

The package also includes UT DVR libraries that enable the extraction of both text and image data from the movie file into 32-bit versions of Matlab on the PC or MAC (tested on version R2010a) and into Vizard on the PC. The code is not compatible with 64-bit versions of Matlab. Additional instructions, example code, and API commands are stored in the SourceForge repository.

The UT DVR package has been tested on 32-bit Windows XP/7 operating systems with Quicktime Pro version 7. The code is compatible with both the Arrington Viewpoint version 2.9.2.6 as well as Applied Science Laboratories (ASL) Eye-track version 6 (Applied Science Laboratories, Bedford, MA). Although the code is currently only compatible with the Arrington and ASL eye-tracking packages, the open source code allows for...
user-end modification for compatibility with alternative eye-tracking packages. Note that users of non-Arrington eye-tracking systems will also need to install a Matrox brand video-capture card that supports the Matrox Image Libraries API (Matrox, Dorval, Quebec, Canada). This card will receive input from scene output of the ASL control unit to enable compression of the visual image. In addition, an additional connection must be made between the serial-out of the ASL control unit to the serial port 1 of the PC.

Nota tion

The following sections require discussing mathematical relationships between several different quantities. We use a consistent notation for clarity. We represent matrices and frames of reference as bolded, capital letters. We represent vectors as bolded lower-case letters. We represent scalar quantities as light-face Latin letters for distances and Greek characters for angles (except for field of view, which is denoted simply as “fov”). Most vectors and matrices and some scalar values have a subscript indicating the frame of reference. For example, the position of the eye within the eye/screen reference frame is \( e_E = [000]^T \). We use a circumflex to distinguish a unit length direction vector from a point in space. The gaze direction within the virtual world is \( \hat{g}_W \). At times we add additional subscripts to specify a particular scalar element from a vector (e.g., the height of the eye within the world would be \( e_{Wz} \)). We also use the subscript notation to indicate the value of a variable from a particular point in time \( t \) when necessary.

Measures of gaze angle and angular distance

From eye-in-head to gaze-in-world

Many measures commonly used to characterize eye movements rely on assumptions unique to the paradigms in which they were developed. For example, the majority of eye-tracking studies involve fixing the subject’s head in front of a two-dimensional planar viewing surface (e.g., a computer monitor). In this situation, there is a 1:1 mapping between the location of a fixated pixel on the screen and the orientation of the eye-in-head. However, in a more natural situation in which the head is unrestrained, changes in subject gaze orientation often result from coordinated movements of the subject’s eyes, head, and body. One approach to characterizing gaze for a more natural setting is to use the gaze vector within the virtual world’s frame of reference.

Multiple reference frames

Finding the gaze vector requires mathematically projecting from pixel coordinates into canonical coordinates. Properly describing the transformations that relate the pixel coordinates provided by the eye-tracker to coordinates in the virtual environment requires a brief discussion of reference frames (Figure 3). Coordinates and directions are always framed relative to a reference point (origin) and a basis set of direction vectors. Graphics libraries, like OpenGL, commonly treat the eye or camera as the default origin and define coordinates in terms of the eye’s orientation. In OpenGL, the \( x \)-axis is to the right of the eye, while the \( y \)-axis is up, relative to the eye. The eye looks along the negative \( z \)-axis. The programmer parametrically specifies an appropriate field-of-view for the eye that is used by OpenGL to calculate the mapping from a point in the virtual world to pixel coordinates when displayed inside the HMD. Thus, the rendering process may be thought of as a conversion from features in the virtual world to locations within the capture volume, and then to pixel locations on the HMD display. Calculating the gaze-in-world vector reverses this rendering process. It is important, therefore, to be sure that the relationships between the various frames of reference are known and available when processing gaze data.

As mentioned previously, our setup uses a binocular HMD tracked within a motion capture volume. Calibrating the capture volume involves a step that defines the origin and axes of the space. Calibrating the HMD to be tracked also involves defining a reference point on the HMD (typically a point that is equidistant between subject’s two eyes) and a default orientation. Subsequently, the motion-capture system can report changes in the position as movements within the capture volume and changes in orientation using a compact description that details rotations away from the helmet’s default orientation (an issue we will return to later). One can use this information to re-create the location of both the eye and the HMD displays inside the virtual world. Finally, we can calculate the gaze vector, which extends from the location of the left eye within the HMD and through the location of the fixated location of the display in the virtual world.

Frame of reference representations

To move points or vectors between each frame of reference (eye, head, capture volume, and virtual
world), one can use matrices that apply the appropriate spatial transform. Depending on how they are to be used, the relationships between different frames of reference can be represented in several different ways. In graphics applications it is common to use $4 \times 4$ matrices that simultaneously bring a point or vector to the proper orientation within the new space, while also translating the point to its proper location. For clarity, however, we will separate the two steps by using $3 \times 3$ matrices (e.g., $\mathbf{M}_F$ in Equation 1) to bring about rotation, and a separate $3 \times 1$ vector (e.g., $\mathbf{p}_F$ in Equation 1) to bring about the proper translation. Note that the subscript $F$ is a placeholder used to indicate which frame of reference the matrix or point uses.

$$
\mathbf{M}_F = \begin{bmatrix}
m_{xx} & m_{xy} & m_{xz} \\
m_{yx} & m_{yy} & m_{yz} \\
m_{zx} & m_{zy} & m_{zz}
\end{bmatrix},
$$

$$
\mathbf{p}_F = \begin{bmatrix}
p_x \\
p_y \\
p_z
\end{bmatrix}
$$

From pixel coordinates to eye coordinates

Popular eye-tracking suites (e.g., Applied Science Laboratories; Arrington Research; Tobii Technology, Danderyd, Sweden) provide pixel coordinates of the fixated location in screen space with the pixel-coordinate origin in the upper-left corner of the screen, and with a height and width of ($\text{pix}_x$, $\text{pix}_y$) pixels. Here, we discuss how to convert pixel coordinates into an angular measure of gaze location in which an eye centered in front of the screen will have horizontal and vertical visual angles of ($\theta_s = 0$, $\varphi_s = 0$) when looking through the central pixel (($\text{pix}_x/2$), ($\text{pix}_y/2$)). This same center pixel expressed in terms of the angular height and width of the field-of-view and measured in visual degrees along the horizontal and vertical screen axes is ($\text{fov}_x$, $\text{fov}_y$). Rearranging the basic trigonometric relationship $\tan(\text{fov}_x/2) = \frac{\text{pix}_x}{2d}$, yields a distance from the eye to the center of the screen, $d = \frac{\text{pix}_x}{2\tan(\text{fov}_x/2)}$. Given some pixel coordinate ($x$, $y$), the screen-relative visual angles are equal to the inverse tangent of the ratio between the pixel distance from this point and the center of the screen along its respective axis, and the distance of the eye from the screen.

Rearranged for clarity, these angles are

$$
\theta_s = \tan^{-1}\left(\frac{2x - \text{pix}_x}{2d}\right)
$$

and

$$
\varphi_s = -\tan^{-1}\left(\frac{2y - \text{pix}_y}{2d}\right)
$$

However, as mentioned previously, it no longer makes sense to use screen-relative angles within a virtual world if the eye and screen are freely moving.
relative to the target. Because of this, we propose a modified measure that uses the angle between the fixation direction (the gaze vector) and vectors between the eye position and targets of interest in the virtual environment. To this end, we use the relationships between the various frames of reference in play to transform the screen-based pixel-coordinate data delivered by eye-tracking systems into world-based direction vectors.

The rules of similar triangles tell us that we can use the visual angles to transform the pixel-coordinate \((x, y)\) to a point, \(\mathbf{g}_E\), on a fictitious plane one unit in front of the eye in eye-relative coordinates (where directions are defined relative to the screen). Referring to Figure 4, we see that the \(x\)-coordinate of this point is \(g_{Ex} = \tan(\theta_x)\). The formula for the \(y\)-coordinate is equivalent and the \(z\)-coordinate is, trivially, one unit ahead. Substituting in the formulas for \(\theta_x\), \(\varphi_y\), and \(d\) from Equation 2, we end up with the following formula for an eye-relative point on a plane one unit in front of the eye:

\[
g_E = \begin{pmatrix}
\frac{2x}{\text{pix}_x} - 1 & \frac{\text{fov}_x}{2} \\
-\frac{2y}{\text{pix}_y} & 1 & \frac{\text{fov}_y}{2}
\end{pmatrix}
\]  

(3)

The second element appears negated because in pixel coordinates, \(y\) increases downward, but in eye space, \(y\) increases upward. The third element is negative because, as discussed earlier, the gaze direction lies along the negative \(z\)-axis in eye space.

Normalizing this three-dimensional, eye-relative point into a unit vector allows us to treat it like a direction instead of a point. This vector provides the gaze-direction relative to the eye/screen frame of reference: \(\hat{\mathbf{g}}_E = \mathbf{g}_E / ||\mathbf{g}_E||\).

**From eye coordinates to world coordinates**

Once we have calculated the direction of the gaze vector in eye space, we then need to transforming the gaze vector from eye space to HMD space, to capture-volume space, and then finally to world space. Note that some HMDs may increase the overall horizontal field-of-view by rotating the display for each eye away from fronto-parallel by \(\rho\) degrees (e.g., in the NVis SX111, \(\rho = 13^\circ\)), in which case one must take an additional step to ensure accurate placement of the gaze vector within the virtual world. Note that the frame of reference is otherwise unchanged. In the case that one is using an HMD with screens that are not fronto-parallel, one must take the additional step to transform \(\hat{\mathbf{g}}\) into HMD direction space from eye space of multiplying with a rotation matrix to rotate the
To properly position gaze within the virtual world, one must also account for the orientation of the head/HMD. Our motion capture system reports the orientation of the HMD with a unit length quaternion: \( q_C = [q_w, q_x, q_y, q_z] \). When used to represent a spatial orientation, a quaternion encodes a rotation by \( \theta \) around an arbitrary unit axis, \([x y z]\), as \( q = [\cos(\theta/2) x \sin(\theta/2) y \sin(\theta/2) z \sin(\theta/2)] \). Using standard formulas from the graphics community (Horn, 1987), we create a rotation matrix \( M_C \) from this quaternion to find gaze direction within the capture volume coordinates established at calibration time:

\[
M_C = \begin{bmatrix}
q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_z q_w) & 2(q_x q_z + q_y q_w) \\
2(q_x q_y + q_z q_w) & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z - q_x q_w) \\
2(q_x q_z - q_y q_w) & 2(q_y q_z + q_x q_w) & q_w^2 - q_x^2 - q_y^2 + q_z^2
\end{bmatrix}
\]

Finally, we transform to world coordinates. By default the capture volume uses the \( y \)-axis as “up,” but in our virtual environment, “up” is along the positive \( z \)-axis. Other axes are consistent between both eye and world coordinates. The following rotation matrix changes the axes appropriately:

\[
M_W = \begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0
\end{bmatrix}
\]

When all of these matrices are applied in the correct order to the gaze vector in eye coordinates, we get the gaze vector in world coordinates:

\[
g_w = M_W M_C M_H g_E
\]

In some cases it may be useful to frame this vector as a pair of world-relative angles: pitch (\( \varphi_w \)) and yaw (\( \theta_w \)). We define pitch as the angle between a vector and the ground plane with positive angles corresponding to the vector pointing upward. The angle between the forward (positive \( y \)) axis and the projection of a vector onto the ground plane is yaw, with positive angles to the right.Yaw is undefined if the vector is perpendicular to the ground plane. One can extract the individual components of pitch and yaw from the world-based gaze vector:

\[
\varphi_w = \sin^{-1}(g_wz)
\]
\[
\theta_w = \tan^{-1}\left(\frac{g wx}{g wy}\right)
\]

**Absolute angular distance from gaze to a target**

In some situations, one may be interested in the angle between the gaze vector and some target location during fixation or pursuit. We compare the angle between the gaze vector to the vector \( f_w \) from the eye to some point of interest \( f_w \) in the virtual environment (Figure 3). This direction vector is equal to the difference between the target point and the location of the eye in the world frame, normalized to unit length:

\[
f_w = (f_w - e_w) / ||(f_w - e_w)||
\]

Finding the location of the eye within the world \( e_w \) is similar to finding the gaze vector, but with some key
differences because we are working with a point instead of a direction. The first step, finding the position of the eye in HMD space, is straightforward. The eye is at the origin in eye space by definition; for convenience, we calibrate the origin of the HMD to be directly between the eyes. The position of the left eye in HMD space is therefore half of the interocular distance (iod) along the negative x-axis: \( e_H = \begin{bmatrix} 0 \\ -\text{iod}/2 \\ 0 \end{bmatrix} \). Although we encourage individual measurement of the iod, we have found that a constant value (iod = 0.06 m) is generally acceptable for most subjects using our HMD.

Finding the eye within the capture volume requires rotating from HMD space into capture space, and then translating the location of the HMD reference point \((h_C)\) to its current location within the capture volume as given by the motion capture software: \( e_C = M_C e_H + h_C \). The final transformation to world space involves a rotation but no translation because we define our virtual world to use the same origin as the capture volume: \( e_W = M_W e_C \). If the virtual environment did not use the same origin, we would translate by adding in the position of the capture origin relative to the virtual world.

Once both \( \hat{g}_W \) and \( \hat{f}_W \) are known, one may calculate the angle between gaze and an object in the virtual world by taking the inverse cosine of the dot product of the two unit vectors: \( \gamma = \cos^{-1}(\hat{g}_W \cdot \hat{f}_W) \).

**Angular distance along two axes**

In some situations, one might prefer to know the directions along which gaze was offset from the target vector. Here, we present methods for distinguishing between horizontal and vertical components of error within an eye-centered coordinate frame. The horizontal component \( \varepsilon \) is measured along an axis \( \hat{u}_W \) that is parallel to the world’s up axis and perpendicular to the gaze vector. The vertical component of error \( \alpha \) is measured along axis \( \hat{h}_W \), which is parallel to the ground plane but perpendicular to both the vertical axis and the gaze vector (see Figure 6). One calculates these vectors using a cross-product:

\[
\hat{h}_W = \frac{\hat{g}_W \times \hat{u}_W}{|\hat{g}_W \times \hat{u}_W|} \tag{9}
\]

\[
\hat{u}_W = \hat{h}_W \times \hat{g}_W \tag{10}
\]
One can calculate angular distance from the gaze vector to the target vector along these axes by first using the dot product, which has the effect of projecting both the gaze and eye-to-target vectors onto these new axes (for example, $\hat{f}_W \cdot \hat{h}_W$, and $\hat{f}_W \cdot \hat{g}_W$). Subsequently, one can recover the distance between these two projections using the inverse tangent of the ratio between the distance along each of these new axes, and the distance of the target to the gaze vector:

$$\varepsilon = \tan^{-1}\left(\frac{\hat{f}_W \cdot \hat{h}_W}{\hat{f}_W \cdot \hat{g}_W}\right)$$  \hfill (11)

$$\zeta = \tan^{-1}\left(\frac{\hat{f}_W \cdot \hat{a}_W}{\hat{f}_W \cdot \hat{g}_W}\right)$$  \hfill (12)

Note that this measure does not provide information on the orientation of the eye within the head but only the angular difference in world space between the gaze vector and the vector from the eye to the virtual object.

Although this methodology allows one to calculate the distance from gaze to the center of an object, it may also be extended to approximate the distance from gaze to the edge of an object. However, calculating the true distance from the gaze vector to the edge of a virtual object is complicated by transformations due to changes in viewpoint and perspective, which will change an object’s projection onto the two-dimensional view plane. Because there is no clear method for extracting information regarding the pixels that define an object’s edge once it has been projected onto the display screen, edge location must be approximated. For example, one might approximate all objects as spheres. Because spheres are viewpoint independent, changes in viewpoint will only bring about changes in visual size that may be easily be approximated using basic trigonometry. Subsequently, one can subtract the approximated angular size of the object from the measurements of angular distance provided above.

**Classifying gaze patterns**

Although algorithms have been developed for identifying saccadic eye movements and fixations when the head is stationary and the viewing plane is fixed (Nyström & Holmqvist, 2010; Salvucci & Goldberg, 2000), these algorithms do not generalize to conditions that afford the subject the freedom of head and body movements. The head-free situation is further complicated by the vestibulo-ocular reflex (VOR) where subjects maintain gaze upon a fixed location during head movements by counter-rotating the eye. Here, we present some methods for the identification of fixations and saccades that are robust to head movements, and the effects of VOR.

**Data filtering**

Prior to the implementation of algorithms for gaze classification, one will likely need to filter the data to reduce stochastic noise and to eliminate brief signal dropouts (e.g., due to blinks). When working with a 60-Hz infrared eye-tracking system, we have found a 3-unit-wide median filter to be effective for the removal of signal outliers and a 3-unit-wide Gaussian filter to reduce jitter. Similarly, when preparing 120-Hz gaze a velocity signal measured with a video-based eye tracker for the identification of smooth pursuit and VOR, Das, Thomas, Zivotofsky, and Leigh (1996) found positive results from the application of a 7-point median and moving average filters.

It is important to note that by smoothing over sudden changes in gaze velocity associated with a saccade, one may find both an artificial reduction in peak saccade velocity as well as a broadening of the saccade duration. The magnitude of these changes will differ with the characteristics of the underlying signal, as well as parameters of the filtering process. For these reasons and more, we hesitate to assume that any single filtering methodology will produce favorable results across the wide range of eye-tracking systems available, and instead advocate tailoring the methodology and parameters of the filtering process on a system-by-system basis. For a more detailed discussion of relevant filtering techniques, see Nyström & Holmqvist (2010).

**Identifying fixations**

Algorithms for the identification of fixation are typically restricted to the analysis of gaze velocity or higher order information. One can numerically differentiate gaze position to calculate the velocity of the gaze vector within the world frame of reference ($g_{vel}$) as the gaze vector changes across subsequent frames:

$$g_{vel} = \frac{\cos^{-1}(\hat{g}_W \cdot \hat{g}_{W_{(t-1)}})}{\Delta t}$$  \hfill (13)

Subsequently, one can identify fixation periods as those in which gaze velocity is below a fixed threshold. This method can also be improved by adopting empirical constraints. For example, one might require that fixations last for a sustained duration greater than a particular minimum. Similarly, one might or clump together fixations that are separated by interruptions.
too brief to be considered saccades, and that may instead be attributed to tracker-related noise.

Note that as the head rotates, the eye moves around the head’s central pivot. Thus, to maintain fixation, the eye must compensate for both the rotational component of the head ($\phi$), as well as the change in perspective ($\zeta$) due to this slight translation. Consider a circular head of radius $r$ and a target that is distance $d$ from the circumference of the head (Figure 7). If the eye is directly aligned between the center of the head and the target during fixation (an initial visual angle of $\theta$), a head rotation of $\phi$ results in an angular change of $\beta = \phi + \zeta$ where $\zeta$ reflects the component of VOR necessary to compensate for the eye’s translation due to head rotation. Given head rotation, head radius, and the distance between the head and target, this extra component is computed:

$$\zeta = \tan^{-1}\left[\frac{r \sin \phi}{r + d - r \cos \phi}\right].$$

In this equation, the bottom portion of the fraction reflects the contribution of eye translation ($r \cos \phi$) to the magnitude of the vector between the eye and the target ($r + d$). The top portion of the equation reflects the perpendicular component of eye translation.

This effect on the angle between the eye and some target object in world space decreases for targets at greater distances. When $d$ is very small, values of $\zeta$ are larger, and contribute to an appreciable increase in the overall angular change required to maintain fixation. However, if $d$ is large relative to $r$, $\zeta$ approaches zero and contributes little to the total angular change required for VOR.

For example, consider the situation when a subject is fixating an object that is at a distance of 5 m, and directly in front of the head. If fixation is maintained with VOR during a $15^\circ$ counter-clockwise head rotation, the eye will be translated along a circular path around the head’s central pivot, and the gaze vector will have shifted approximately $0.3^\circ$ in eye coordinates relative to a fixed exocentric reference angle (Figure 8). Note that $0.3^\circ$ is likely below the level of noise expected by most eye-tracking devices. The effects are more pronounced when the object is closer to the head. For example, when fixating an object 0.5 m away from the eye, the same $15^\circ$ head rotation would bring about approximately $3^\circ$ of shift in eye-centered coordinates.

### Identifying gaze shifts

To identify coordinated movements of the eye and head, we adopt methods based upon those of Duchowski et al. (Duchowski, Medlin, Cournia, Gramopadhye et al., 2002; Duchowski, Medlin, Cournia, Murphy et al., 2002), which involve convolving the eye-in-head signal with a kernel that is loosely representative of a paradigmatic gaze shift in both duration, and in shape. For example, Duchowski et al. adopted the kernel $[0 \ 1 \ 2 \ 3 \ 2 \ 1 \ 0]$. When applied to a 60-Hz signal, this 7-unit filter is representative of a 5-unit bell-shaped profile that spans a duration of approximately 83 ms, and that is flanked by two periods of inactivity.

Convolution with gaze velocity produces a filtered signal in which stochastic noise is diminished. Although movements of the head may introduce saccadic signals that do not conform to the bell-shaped filter, the filter still proves effective, likely because high-amplitude saccadic eye movements are easily distinguished from low-velocity head movements and VOR (Duchowski, Medlin, Cournia, Gramopadhye et al., 2002).

Although the methods proposed by Duchowski et al. are specific to the identification of the time at which a
gaze shift reached peak velocity, one can easily extend these methods to facilitate identification of both the beginning and end of each gaze shift. Using a modified kernel in which the profile is flanked by negative values (e.g., \([-1 \ 0 \ 1 \ 2 \ 3 \ 2 \ 1 \ 0 \ -1\]) produces exaggerated valleys in the filtered gaze velocity signal just before and after the gaze shift, as is seen in Figure 9. These valleys can be exploited during the identification process.

The process begins with the identification of saccade peaks using a fixed threshold on the filtered velocity signal. Note that the saccade peak is accompanied by a zero crossing of acceleration, as seen in the lower panel of Figure 9. This zero-crossing is flanked by two nonzero changes in acceleration direction at the time of the saccades start and end that are identifiable as a local minimum/maximum. The starts and ends of three saccades are identified as the frame before and after these local minima/maxima, as depicted by the shaded regions in Figure 9. Note that, due to signal broadening during the filtering process, the saccade duration may be slightly elongated with respect to the raw signal (dotted line). This issue is discussed briefly in the previous section on data filtering.

### Identifying tracking via pursuit eye movements

In addition to fixation and saccadic eye movements, subjects may engage in smooth pursuit of a moving object. Pursuit eye movements are often studied using a step-ramp paradigm in which a target appears several degrees away from the subject’s initial fixation point and moves in the opposite direction so that it crosses over the location of the initial fixation point during smooth pursuit. Within this context when the head is unrestrained, it takes approximately 140 ms after the initial step for subjects to initiate smooth pursuit to a constant velocity target (Ackerley & Barnes, 2011). The quality of pursuit is typically characterized by initial period of gaze acceleration that causes the ratio between pursuit velocity and object velocity, or pursuit gain, to approach unity. Although it becomes increasingly difficult to sustain unity gain at higher velocities, pursuit gain as high as 0.9 has been observed for targets moving up to 100°/s when the head is fixed (Meyer, Lasker, & Robinson, 1985), and at speeds of 184°/s when the head is unrestrained (Hayhoe, McKinney, Chajka, & Pelz, 2012).

However, research on gaze in natural environments has demonstrated that tracking of an object moving through natural environment is typically accomplished using a combination of smooth pursuit and catch-up saccades (Barnes, 2008; Orban de Xivry & Lefèvre, 2007). These saccades occur in response to a large angular distance between gaze and the object, or periods of poor pursuit characterized by low pursuit gain and serve to bring about a net reduction in the distance between gaze and the moving target. By identifying saccades that meet these criteria, one can clump periods of pursuit that are interrupted by catch-up saccades into a single instance of coordinated object tracking. In the following section, we present a method for the calculation of head-free pursuit gain to be used for identifying both periods of smooth pursuit and tracking using a combination of smooth pursuit and catch-up saccades.

### Calculating pursuit gain

To compute the angular velocity of the target around the eye (\(\omega_T\)) we measure the angular distance between the vector across frames of time. The cross-product between the vector recorded at two different points in time results in a new vector, \(c_T\), orthogonal to both measured vectors, scaled by the sine of the angle between them. Normalizing this vector and then scaling by the angle gives the angular velocity in radians per frame around some axis in the world frame.
We find pursuit gain by projecting vector \( \mathbf{f}_{t} = \hat{\mathbf{w}}_{t} - \hat{\mathbf{w}}_{t-1} \times \hat{\mathbf{f}}_{t} \) \( (15) \)

\[
\omega_{f_{t}} = \frac{\mathbf{f}_{t} \sin^{-1}(||\mathbf{f}_{t}||)}{||\mathbf{f}_{t}||} \quad \text{with } \omega_{f_{t}} := 0
\]

when \( ||\mathbf{f}_{t}|| = 0 \) \( (16) \)

We then compute the angular velocity of the gaze vector \( \mathbf{g}_{t} \) within the world using the same formula. We find pursuit gain by projecting \( \mathbf{g}_{t} \) onto normalized vector \( \mathbf{f}_{t} \) and then dividing by the magnitude of \( \mathbf{f}_{t} \):

\[
\beta = \frac{\langle \mathbf{g}_{t} \cdot \mathbf{f}_{t} \rangle}{||\mathbf{f}_{t}||}
\]

This same computation could be done with vectors in the HMD frame of reference to compute pursuit gain without head movement.

There is a singularity in this computation when \( \mathbf{f}_{t} = 0 \). However, if the angular velocity of the target around the eye is zero, then pursuit gain might not be an appropriate measure since pursuit implies a moving target. In this case we can still compute the change in absolute angular error between the gaze and the target over time although this is not the same as pursuit gain: \( \gamma_{t} - \gamma_{t-1} \).

To identify periods of smooth pursuit, one may search for durations during which pursuit gain lies within high and low boundary thresholds, and angular gaze-to-target distance lies below a maximum threshold value. We suggest threshold gain values of 0.3 and 1.2, and a maximum distance of 5° may be suitable for the identification of pursuit when using a video-based infrared eye tracker, as in Diaz et al. (2013). Methods for identification of smooth pursuit may also be extended for the classification of tracking behavior with combined pursuit and catch-up saccades. To identify catch-up saccades, one must first identify interruptions to smooth pursuit. Subsequently, one may apply the algorithm for saccade detection to ensure that this period was accompanied by a saccade, and examine change in angular distance before and after the saccade to ensure that this saccade brought about an overall decrease in gaze-to-target distance.

It is worth mentioning here that these measures can be quite sensitive to discrepancies in the temporal alignment between eye-tracking data, motion-capture data, and data from the virtual environment such as target locations. This is one reason that the digital video recording software, such as the UT DVR, is important for subsequent validation of computational methods.

Our methods build upon those of Duchowski, Medlin, Cournia, Gramopadhye, et al. (2002) for the identification of fixations and saccadic eye movements in a virtual environment. We also present methods for the calculation of gaze-in-world angles, angular distance from gaze to an object in the virtual world, and algorithms for the identification of pursuit eye movements. To aid in temporal alignment and of gaze data, numerical data about state of the virtual world, and to provide a means for visual inspection of the in-helmet view presented to the subject, we have provided the UT DVR libraries. Using these tools, investigators investigate the allocation of gaze within a virtual environment, in which the head and body are unrestrained.

Although, these methods are specific to the analysis of monocular eye-tracking data, it is possible that they can be extended to the analysis of binocular eye-tracking data by treating the binocular signal as an averaged combination from each individual eye. The subsequent process of analysis would likely be preceded by the estimation of a single gaze vector that takes into account the combined gaze directions and depth of gaze, as is indicated by the point at which the two individual vectors are closest. It remains unclear, however, how tracker noise will effect these measurements at appreciable distances, or how the additional information about depth-of-gaze provided by a binocular tracker should be combined with gaze direction to provide an estimate of fixation location in a three-dimensional environment.

**Keywords:** virtual reality, eye movements, gaze, methods

---

**Acknowledgments**

Our thanks to John Stone for his considerable efforts programming the UT DVR libraries, and our anonymous reviewers for their insightful comments. This research was supported by NIH grant EY05729.

Commercial relationships: none.
Corresponding author: Gabriel Jacob Diaz.
Email: gdiaz@mail.cps.utexas.edu.
Address: Center for Perceptual Systems, University of Texas Austin, Austin, Texas.

---

**Footnotes**

1www.nvisinc.com
2www.phasespace.com
3www.arringtonresearch.com


