libGaze: an open-source library for estimating the gaze of freely moving observers in real-time

|  |  |  |
| --- | --- | --- |
| Sebastian Herholz | Max Planck Institute for Biological Cybernetics,  Tübingen, Germany | [mail_icon_2](mailto:lewis.chuang@tuebingen.mpg.de) |
| Lewis L. Chuang | Max Planck Institute for Biological Cybernetics,  Tübingen, Germany |  |
| Thomas G. Tanner | Max Planck Institute for Biological Cybernetics,  Tübingen, Germany |  |
| Roland W. Fleming | Max Planck Institute for Biological Cybernetics,  Tübingen, Germany |  |

Most eye-tracking systems require the users’ head to remain stationary during tracking. This restricts the tasks that can be performed by the subject, and can lead to unnatural gaze movements, especially when the field of view is large. Here we present a practical end-to-end system for tracking the gaze of an observer in real-time as they move around freely and interact with objects or a large display screen. The core of the system is a software library, LibGaze, which combines data from off-the-shelf eye- and body-tracking hardware to estimate the users’ gaze in 3D space in real-time. Due to its modular design, key components of the system (e.g., eye-tracking hardware or calibration algorithm) can be easily substituted, making the system hardware independent. In a series of experiments we evaluate the accuracy and stability of the system, and describe some common sources of error, along with practical guidelines for ensuring good tracking. We also provide a detailed description of how to incorporate LibGaze into experiments, including example code. Although previous work has described calibration algorithms for estimating gaze this way, here the emphasis is on a flexible, readily available implementation that can be easily adopted by researchers.

Keywords: eye movement, calibration, real-time gaze, mobile gaze tracking, algorithm

Introduction

Under natural viewing conditions, humans typically direct their gaze at points in the surroundings using a combination of eye, head and body movements [Hayhoe & Land, 2005]. By contrast, when tracking gaze in the laboratory, it is common to restrict head and body movements using a bite-bar or chin-rest. While restraining the head can increase the accuracy and stability of eye tracking, it limits the tasks that the subject can perform, and reduces the range of possible gaze movements from about 260° to 110° (Guitton & Volle, 1987; Chen, Solinger, Poncet & Lantz, 1999). Furthermore, the movement kinematics of unrestrained gaze differs from the main sequence that is consistently replicable in restrained eyetracking (Freedman, 2000). Therefore, Of equal concern, is the possibility that restrained eyetracking could introduce behavioral artefacts in gazetracking experiments; for example, contributing to the central bias that is often noted in eye-movement studies (Tatler, 2007). Given these constraints, we argue that there are many circumstances in which it would be desirable to allow the subject to move freely while tracking their gaze.

Several approaches exist for unrestrained gaze tracking. [We need to say something about the commercial systems mentioned in the Johnson et al paper!] While the calibration algorithms for computing this 3D gaze vector have been developed (Ronnse, White & Lefevre, 2007; Johnson, Liu, Thomas & Spencer, 2007), fully implementable systems are not readily available. Studies of unrestricted gaze behavior often utilize setups and systems that are highly specific to their experimental design, such as a {desribe system} (Epelboim, 1997). Alternatively, they might be unsuitable for prolonged testing, such as the scleral search coil method (Freedman, 2000, 2008; Zangemeister & Stark, 1981) or involve arduous handcoding by video-analysis (Land, 1999, 2004). More recent advances in unrestricted gazetracking technology have simply done away with computing the gaze vector and simply records a video from the user’s point of view (Schumann et. all., 2008). This is not ideal for studying how humans coordinate eye and head movements in gaze control [EXPLAIN WHY].

Remote systems that combine markerless head-tracking and eyetracking remove the need for headgear and provides data for both head and eyemovements. However, state of the art systems continue to have high latencies of up to 50 msec and a limited coverage (i.e, 50° of head rotation, at distances not more than 1.4m from the cameras). This renders such systems useless for realtime gaze-contingent displays and interfaces, wherein even small latencies can result in sluggish performance or the employment of unnatural cognitive strategies (Gray & Boehm-Davis, 2000).

In this paper, we present a practical, end-to-end system for tracking eye and head and body movements in real time, while the observer moves freely. The core of the system is a software library, **libGaze**, that combines data from a head-mounted video-based eyetracker and a body motion capture (MoCap) system, to compute online estimates of the user’s gaze. LibGaze also contains calibration functions and commands for controlling data output, to make it easy for the researcher to integrate the system into experiments.

For the purposes of our system, we define gaze as a 3D vector that represents the point from which gaze originates and the direction that it is pointed towards. For simplicity, we assume an equivalent gaze origin for both eyes, located on the nasal bone between them. Hence, the output of out system is a 3D vector which **jointly represents the position where the participant’s gaze originates (e.g, the left eye) and its direction.** This output is continuously updated at a latency of 10msec from time of recording and is described within a world coordinate system (WCS); that is, the physical room in which the system is installed, or a virtual world that is presented to the observer. Given a 3D model of the surface(s) for which gaze position is to be estimated (e.g. the projection screen, or objects in the scene), the system is also capable of deriving the screen coordinates of the POR in realtime. libGaze is written in the C programming language, is universal for hardware types and contains built-in functions for calibration procedures. It has multiple application programming interfaces for the programming languages of: Java, Python, C#, C++. This means that it integrates easily with popular means of experimental control (e.g., VisionEgg (Straw, 2009) or PsychoPy (Pierce, 2007, 2009)) and allows for tracking unrestrained gaze on large displays (e.g., 1.8m by 2.0m).

The primary purpose of this article is to report a gazetracking system that can compute user’s gaze in realtime and carry out accurate calibrations. In the next section, we will describe the required hardware and the calculations that have to be performed in order to compute gaze in the world coordinate system (WCS), given data from a bodytracking (MoCap) and eyetracking system. Similar computations have been described in more detail elsewhere (Ronnse, White & Lefevre, 2007; Johnson, Liu, Thomas & Spencer, 2007) and hence, we will only describe our algorithm briefly. The focus will be on the actual implementation of this system, especially with regards to a software library (libGaze) that we have made publicly available (www.sourceforge.net). More specifically, we will discuss how libGaze is implemented using its Python API (i.e., PyGaze). We have chosen to do so because Python is a non-proprietary programming language, with well-developed libraries for experimental control (e.g., VisionEgg (Straw, 2009) and PsychoPy (Pierce, 2007, 2009)) and a popular base among vision researchers. As a rule, example code will accompany our explanation of how to utilize the library to compute 3D gaze, perform calibrations and log relevant data. These are excerpts from a full demo script, provided in Appendix A. Finally, we will report a full evaluation of this system’s robustness and accuracy to allow the reader to assess its viability. The overall goal is to encourage implementation this gazetracking system, which allows for realtime unrestrained gaze to be accurately computed.

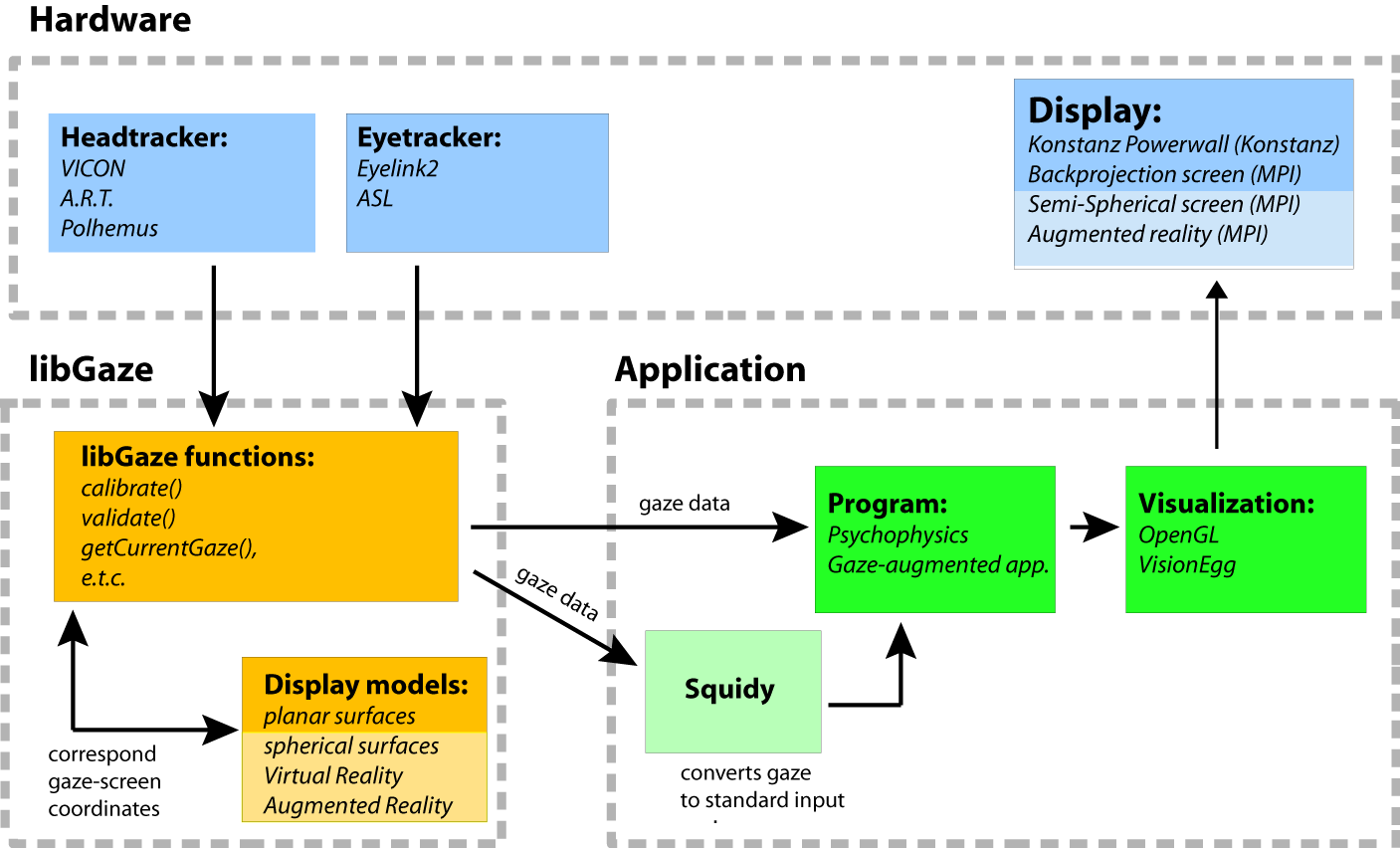


Figure : A schematic representation of the real-time gaze tracking system that is presented in this paper.

System description

Commercial systems for eyetracking are not designed to track unrestrained gaze over a large field of view. Therefore, most experiments are conducted in a head-restrained setting. To track gaze in a 3d world coordinate system, it is necessary to track both head and eye movements. Hardware systems for body motion tracking are commercially available but do not integrate easily with commercial eyetracking systems. libGaze is a software library that coordinates the data from eyetracking and bodytracking hardware as well as contain functions for calibrations and gaze computation in realtime.

Hardware and coordinate systems

[eyetracker, motion capture system, router, visual display.]

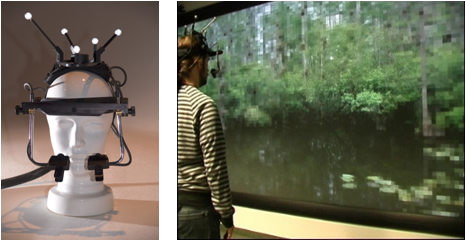
Video-based eyetrackers typically employ head mounted high speed cameras (90-500 Hz) to image the observer’s eyes. They track the pupil in the camera image across time and return the (x,y) screen coordinates of the pupil’s centroid. With an appropriate mapping function (see below), these data can be transformed into spherical coordinates, in an **eye coordinate system (ECS)**, to denote the rotations of the eye in head. We use the Eyelink2 (SR Research).

MoCap systems typically employ infrared cameras (~120Hz) to track reflective markers in a circumscribed space, in the WCS. It would be ideal to track the position of the eye directly, to obtain the origin of gaze. However, this cannot be achieved without obstructing vision. Hence, we track a fixed marker on the eyetracker instead (at the top of the observer’s head). This marker is treated as the origin of a **head coordinate system (HCS)** and the positions and orientations of the eyes within this HCS is measured prior to experimentation during the head calibration (see pg. 5). Knowing this stable relationship of the eye(s) to head marker allows libGaze to compute the position and oriented direction of the eye in the WCS.

The observer’s POR on a large display can also be estimated if a 3D model of the display is provided. This is the intersection point of gaze and the display-plane. Our display setup comprises a JVC projector with large backprojection screen (192cm x 218cm). The display dimensions are 174cm x 215cm, with a resolution of 1280 x 1024.

Estimating gaze origin and direction

The video-based eyetracker returns the screen coordinates of the pupil’s position in its camera recordings. Using established calibration procedures (see. pg. 5), a mapping function can be derived to transform such readings to a direction vector (*eyetracker*) that represent rotations of the eye in head; that is, within the ECS. Still, further transformations will have to be applied to it to obtain a gaze vector in the WCS.

The MoCap system continuously reports a 3D vector in the WCS that specifies the position and orientation of a configuration of retro-reflective markers (*bodytracker*), which is attached to the eyetracker. We assume a rigid relationship between this *bodytracker* and the eyes, within a HCS that treats the *bodytracker* as its origin. This relationship is measured prior to experimentation (see pg. 5). From this, it is possible to determine the translation vector and the rotation matrix, that are necessary transform a vector in the ECS to the WCS. This allows the system to continuously compute the gaze vector of a participant, in the WCS, during experimentation. More details are provided in the subsequent section that explain the use of libGaze (see pg. 5)

~~The MoCap system returns (i) a 3D vector, ~ , which specifies the position of the eyetracker in the WCS, and (ii) a 3x3 matrix, O that describes the orientation of the eyetracker in WCS. The relationship between the tracked object and the observer’s eyes can be described in terms of a translation vector, ~vHCS he and a rotation matrix, RHCS he that together represent the transformation from HCS to ECS. As the position of the eye relative to the head is assumed to be constant, this mapping is measured only once, in the calibration procedure.~~

Figure . left: Eyetracker (SR Research) with retro-reflective markers (VICON) for headtracker. right: Multiresolution wall-sized display that selectively renders high-resolution graphics at the current point-of-regard

~~~vWCS g = ~pWCS h + OWCS h \_ ~vHCS he (1)~~

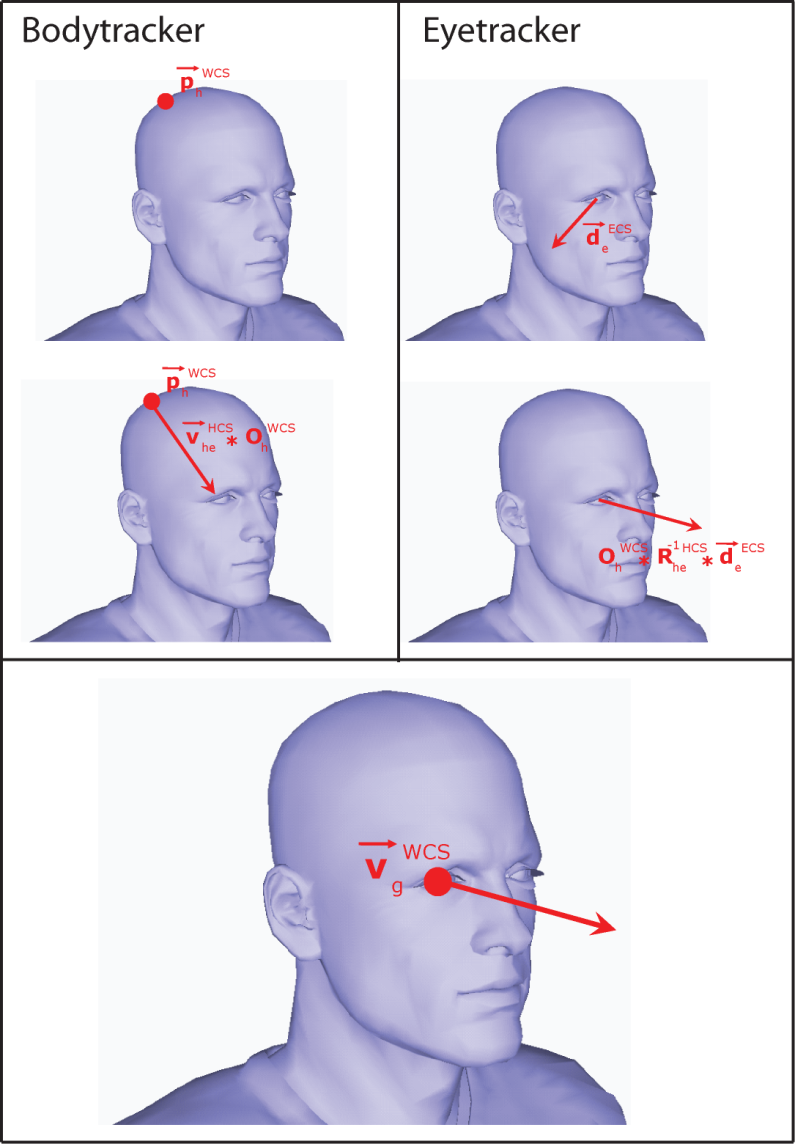
~~In our setup, the eye-tracking system returns the screen coordinates (x, y) of the pupil’s centroid in the camera images. By using a mapping function M(x, y) as described by [7] the 2D image position can be mapped to a 3D viewing direction vector for the eye ~dECS e in the ECS. This mapping function is estimated from the calibration procedure, described below.~~

~~~dECSe = M(x, y) (2)~~

~~To translate ~dECSe into the gaze direction ~dWCS g in WCS, we first translate it into the HCS. From there it can be easily translated into the WCS using the orientation matrix OWCS h of the head tracked object. To translate ~dECSe to the HCS, ~dECS e has to be multiplied with the inverse of the rotation matrix relating HCS to ECS.~~

~~~dWCSg = OWCSh\*RHCShe-1\* ~dECS e (3)~~

~~Given ~vWCSg and ~dWCSg , it is possible to compute the intersection of the gaze ray with any other known surface, as long as it’s physical dimensions in the WCS is known.~~

Software library

A software library (libGaze) handles data from the tracking systems and coordinates them for calculating realtime gaze. This software is implemented in a platform independent library, written in the programming language of C, with application programming interfaces (APIs) for the languages of Java (JGaze), Python (PyGaze), C++ (libGaze++) and C#(csGaze). In this paper, we shall focus on the PyGaze API.

libGaze was created to be universal for hardware types and have been tested with 2 different eyetrackers and MoCap systems. To be independent of specific hardware, libGaze uses a modular system to wrap its underlying hardware components. There are four different module types and each module has a different task as well as a specific set of functions. Each module is implemented as a dynamic C library, loaded at run-time. These modules are described here and Appendix B contains a more detailed overview of the most commonly used functions.

Eye-/Head-tracker module/class

An eye- or head-tracker module acts as a driver for the tracking system, used to track the eye- or head movements. The driver must implement functions for: opening a connection with the tracker system; disconnect; starting and stoppin gthe tracking process; and getting the current tracking data from the tracker.

Display module

The content can be presented to the observer on a range of display types, including large planar projection walls; tiled displays; curved screens or display cubes. This flexibility is achieved by out-sourcing the calculation of gaze-position for each display type to a display module. Each display module offers libGaze a set of functions for calculating 2D display coordinates of current gaze (POR) as well as return the 3D position of a 2D display coordinate in WCS.

An auxiliary program (DisplayCalib) is written for producing a 3D model of any planar displays for use with the display module of libGaze. This 3D model allows libGaze to estimate current gaze (and head) orientation in terms of screen coordinates. In turn, this allows a visual stimulus to be displayed in terms of the amount of eye and/or head movement that is required of current gaze for the stimulus to be fixated. As we shall see, this is essential for the calibration of this gazetracking system.

The display model that is generated from this procedure simply denotes the shape, size and orientation of the surface upon which visual stimuli will be rendered. It is typically described in terms of the WCS and can be in any position, relative to the observer. Hence, the display model can be generated to suit the experimental purpose. For natural scene viewing studies, the display model might be designed to be a standard upright screen. Alternatively, it could be modeled as a tabletop display for pointing studies.

Figure Raw data from the bodytracker (left) and eyetracker (right) are processed with measured transformations and combined to yield a gaze vector.

Calibration module (see next section)

The mapping from 2D pupil position to a 3D gaze vector can be performed by different mapping algorithms, which differ in terms of accuracy and stability. Here, we use Stampe’s (1993) algorithms but the modular architecture of libGaze allows different algorithms to be easily implemented. The calibration module offer functions to calculate the mapping function; to calculate the gaze vector from 2D pupil positions in realtime; and to apply a drift correction to the calculated mapping function. The experimental procedure for performing calibrations is described in the next section.

Using libGaze

GazeTracker object

The main functionalities of libGaze are encapsulated in a gazetracking object (labelled ‘*gt’* in the example code) that has to be created at initiation of experimentation. During experimentation, this object loads the required modules, integrates data across the different modules and allows execution of the necessary calibration procedures.

*# creating an GazeTracker object that prints out # debug and error commands*

gt = pyGaze.GazeTracker(1,1)

*# loading eyetracker, headtracker and display   
# modules as well as set a mode for accepting   
# tracked data from both eye- and head-tracker*  
gt.loadModules(eyemod,headmod, pyGaze.PG\_EHT\_MODE\_BOTH)

*# configuring the eyetracker, headtracker and   
# display modules using configuration files*

gt.configure(eyecfg, headcfg)

Calibration procedures

A system calibration has to be conducted, prior to experimentation, before gaze vector can be computed. These calibrations determine:

1. the position and orientation of the eye, relative to the rigid marker which is tracked by the MoCap system and attached to the top of the eyetracker (Figure 2, top panel)
2. the function that maps the pupils’ screen position on each eyetracking camera and their corresponding rotation in the ECS (Figure 2, middle panel)
3. the dimensions of the display model in the WCS

In addition, a corrective procedure is periodically conducted (e.g., every 15 minutes) to compensate for drift errors, introduced by eyetracker slippage.

Estimating the eye position in HCS

It is necessary to know the relationship between the object that is tracked by the body tracking system and the observer’s eyes. Because this relationship is assumed to be relatively stable throughout an experimental session, the calibration to determine this is only performed once and only at the very beginning.

Prior to any experimentation, the position of the eyes is measured with the aid of an additional tracked object, using the MoCap system. Taking minimal errors into account, the position of both eyes is assumed to be at the nasal bridge. Hence, the observer is instructed to place the tip of a tracked wand on the nasal bridge. The relationship between the MoCap tracked object ~pWCSh and the wand ~pWCS e is then recorded. Because both points are represented in the WCS, the translation vector has to be transformed into the HCS:

~vHCShe = OWCSh−1\_ (~pWCSe − ~pWCSh ) (4)

*# computes the position of the eye relative to   
# the tracked body marker*

gt.collectEyeHeadRelation()

Estimating the orientation of ECS relative to HCS

The relationship between the orientations of the HCS and the ECS is represented by the rotation matrix RHCS he , which is estimated using the following calibration procedure. The subject is asked to assume a comfortable neutral head-pose with gaze straight ahead. The subject is then presented with a fixation point in the center of a large (50◦ by 40◦) rectangular frame - representing the observer’s field of view (FOV) - whose position and orientation is adjustable. The position of each corner of the rectangle is calculated using the current eye position and a predefined viewing direction in ECS multiplied by a combination of the rotation matrix RHCS eh (initialized with the rotation angles X,Y,Z = 0.0◦) and the current head orientation matrix, OWCSh . The orientation of the FOV rectangle is manually adjusted by changing the rotation angles of ~RHCShe until the fixation point is in the center of the observer’s field of view, and the top and bottom of the rectangle is perceived to be horizontal by the observer.

*# displays a rectangle of dimensions 50° x 40°   
# that the user reorients to realigns to current  
# field of view*gt.correctHeadDirectionVector(50,40)

Calibrating the eyetracker

The mapping function M(x, y) is fitted using a standard procedure for calibrating video-based eye-trackers [Stampe93, Moore96]. The goal is to map the 2D coordinates of the pupil’s centroid in the camera images provided by the eye-tracker into a 3D viewing vector in ECS, represented with two angles using spherical coordinates.

During calibration, the observer is presented with a sequence of fixation points. Each fixation point is rendered so as to require the participant to rotate his eyes in a pre-specified direction, by a specific amount, after taking into account the participant’s current head position in WCS. In other words, these fixations points are rendered according to their positions in the participant’s field of view. Altogether, they describe a grid on the participant’s field of view. Each fixation point is displayed until a stable fixation is achieved for 500 secs and the camera position of the pupil is recorded. This is conducted for all the fixation points on the calibration grid and from this, a mapping function is derived that fits the angular rotation of the eyes with their respective camera positions. This fit is then validated by repeating the fixation grid sequence, and measuring the angular error between the estimated gaze position and the true position of the fixation points. If the mean error is below a user defined threshold (e.g. 1.5°) the calibration is accepted, otherwise the calibration procedure is repeated.

*# calculate a mapping function between the   
# video image and eye movement amplitude by   
# presenting a random sequence of evenly   
# separated fixation dots, which are drawn from # a 3 x 3 grid of the dimensions 50° x 40°.*

gt.calibrate(50,40,3,3,None)

*# validate the mapping function by repeating the # same procedure as above*

*# create validation dataset object and run   
# validation procedures*

vDataSet = pyGaze.ValidationDataSet()

gt.validateCalibration(50,40,3,3,None,vDataSet)

*# display validation results onscreen by drawing   
# fixation positions of validation position   
# relative to the grid*

gt.displayLastValidation()

*# print the following data contained in   
# ValidationDataSet to a string   
# "ValidationDataSet: num: ",vDataSet.num,   
# "\tgood: ",vDataSet.good,"\tbad:",  
# vDataSet.bad,"\tavg\_drift:",  
# vDataSet.avg\_drift,"\tmax\_drift:",  
# vDataSet.max\_drift*

print "ValidationDataSet: ", vDataSet.\_\_str\_\_()

Drift correction

This is an additional procedure to intermittently check if the calibration of the eye-tracker is still valid and ~~to correct for small drifts that accumulate over time~~, as a result of equipment slippage. This is especially important for our mobile system as free head and body movements can cause larger errors to accrue, compared to standard head-restrained setups. In the accompanying code, three fixation points are presented in sequence: at the center of the estimated field-of-view, approximately 32° away from the center in one of four possible diagonal directions, at the center of the field-of-view again. If the mean of the errors between the measurements taken at these points and their estimated positions is less than 2.0°, the collected data can be used to adjust the eyetracker mapping function. Otherwise, a full recalibration is advised.

# create an object to contain drift correction   
# data

dcDataSet = pyGaze.ValidationDataSet()

*# perform drift correction with the center point   
# (screen coordinates: 640, 512) of a grid of   
# the dimensions (50° x 40°) with 3 x 3 fixation  
# points. Only 2 points are presented for drift  
# measurement.*

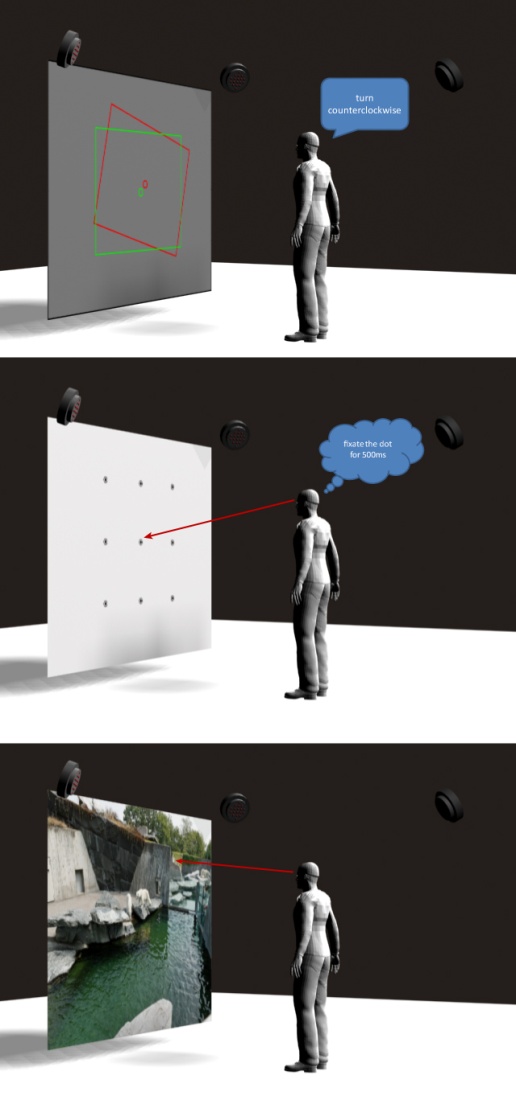
gt.driftCorrection(500,500,50,40,3,3,2,dcDataSet)

Figure 4. System calibration. Top panel: An estimated field-of-view (red) is projected on the display, after the WCS position of the eye is recorded. This projection is adjusted until it is aligned with the observer’s subjective field-of-view (green). Middle panel: A standard calibration of the eyetracker is performed by requiring the observer to fixate points with known positions within the estimated field-of-view. Bottom panel: The intersection of a calibrated observer’s gaze with a large display can be computed in real-time.

*# print the drift correction data to a string   
# object.*

print "DriftCorrectionDataSet: num: ",dcDataSet.num, "\tgood: ",dcDataSet.good,"\tbad: ",dcDataSet.bad,"\tavg\_drift: ",dcDataSet.avg\_drift,"\tmax\_drift: ",dcDataSet.max\_drift

Data output

libGaze offers a default set of data output (*gds*) for post-processing, with options for additional data. The operator is able to determine whether data from both eyes are required or if libGaze should only provide data from the best-calibrated eye. Data is logged at a sampling rate of the slowest tracking hardware, typically the MoCap at 120Hz. Messages can also be inserted into the data output to log experimental events e.g., stimuli onset times and parameters. Table ? provides an overview of the data types and associated functions for logging them.

In real-time, the operator can call for the current gaze vector with the function ?? from the gazetracker module. This can also be translated into the current POR, in terms of display coordinates.

In addition, a parser (written in Python) allows this dataset to be converted offline, to yield only the relevant data, according to the needs of the experimenter.

(DISPLAY\_GAZE): ldis\_x ldis\_y rdis\_x rdis\_y

(GAZE): lgorigin\_x lgorigin\_y lgorigin\_z lgdirection\_x lgdirection\_y lgdirection\_z rgorigin\_x rgorigin\_y rgorigin\_z rgdirection\_x rgdirection\_y rgdirection\_z

(EYE): leye\_a leye\_b leye\_size reye\_a reye\_b reye\_size

(DISPLAY\_HEAD): hdis\_x hdis\_y

(HEAD): hpos\_x hpos\_y hpos\_z heuler\_x heuler\_y heuler\_z

*# data available in gaze data*

print "GazeDataSet: gaze\_available: ", gds.gaze\_available, "\teye: alpha:", gds.eye.alpha,"\teye: beta:", gds.eye.beta,"\teye: size:", gds.eye.size, "\thead: position: ", gds.head.position, "\thead: orientation: ", gds.head.orientation

print "GazeDataSet: gaze[0]: position", gds.gaze[0].position, "gaze[0]: direction", gds.gaze[0].direction,"gaze[1]: position", gds.gaze[1].position, "gaze[1]: direction", gds.gaze[1].direction

*# calculating screen position of gaze on a   
# display*

*#creating a Display object*

display = pyGaze.Display()

*#loading a specific model of displays*

display.loadModule(dismod, 0,1)

*#configuring the loaded display model*

display.configure(discfg)

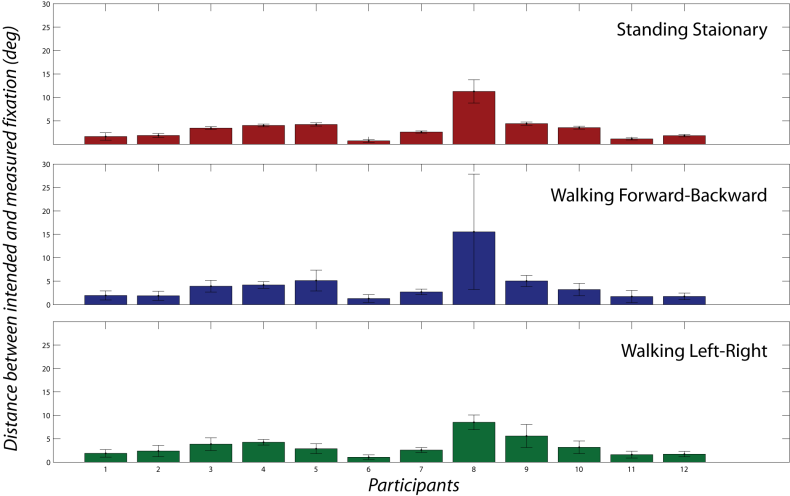
*# calculates and prints out the intersection of # current gaze and display in screen coordinates*

dc = display.getDisplayCoordsFromGaze(gds)

print ...  
"DisplayCoords:",dc.x,dc.y,dc.avg\_x,dc.avg\_y

Evaluation and Results

The primary goal of this system is to enable a laboratory environment for natural gaze-tracking that met the following criteria:

1. Robustness to free body movement
2. Accurate calibration method that generalized across an unlimited viewing space
3. Accuracy in natural gaze measurement

For this purpose, a full evaluation was carried out to test the system. The following procedures were designed to assess the overall error, contributed by equipment slippage, hardware and algorithm, that were present in our implementation of libGaze. All participants were young adults, (age: 25-35 years) with normal or corrected-to-normal vision.

In all of these evaluations, the difference between calculated fixation position on the screen and the stimulus display coordinates is used as a measure of error in our gaze estimates. These errors are unlikely to have resulted from our algorithms for gaze computation and instead, reflect tracking errors due to headgear slippage, limitations in the spatial resolution of our tracking hardware, variable sampling frequencies and individual differences in gaze accuracy. We report these results for three purposes. First, as a benchmark for future improvements to the system. Next, as a way of assessing independent implementations of our system. Finally, for researchers to assess this system’s suitability for their respective research.

Figure . The mean distance of each participants’ measured gaze to the ideal gaze vector (°) when they are standing stationary (top), walking forward-backward (middle), and walking left-right (bottom).

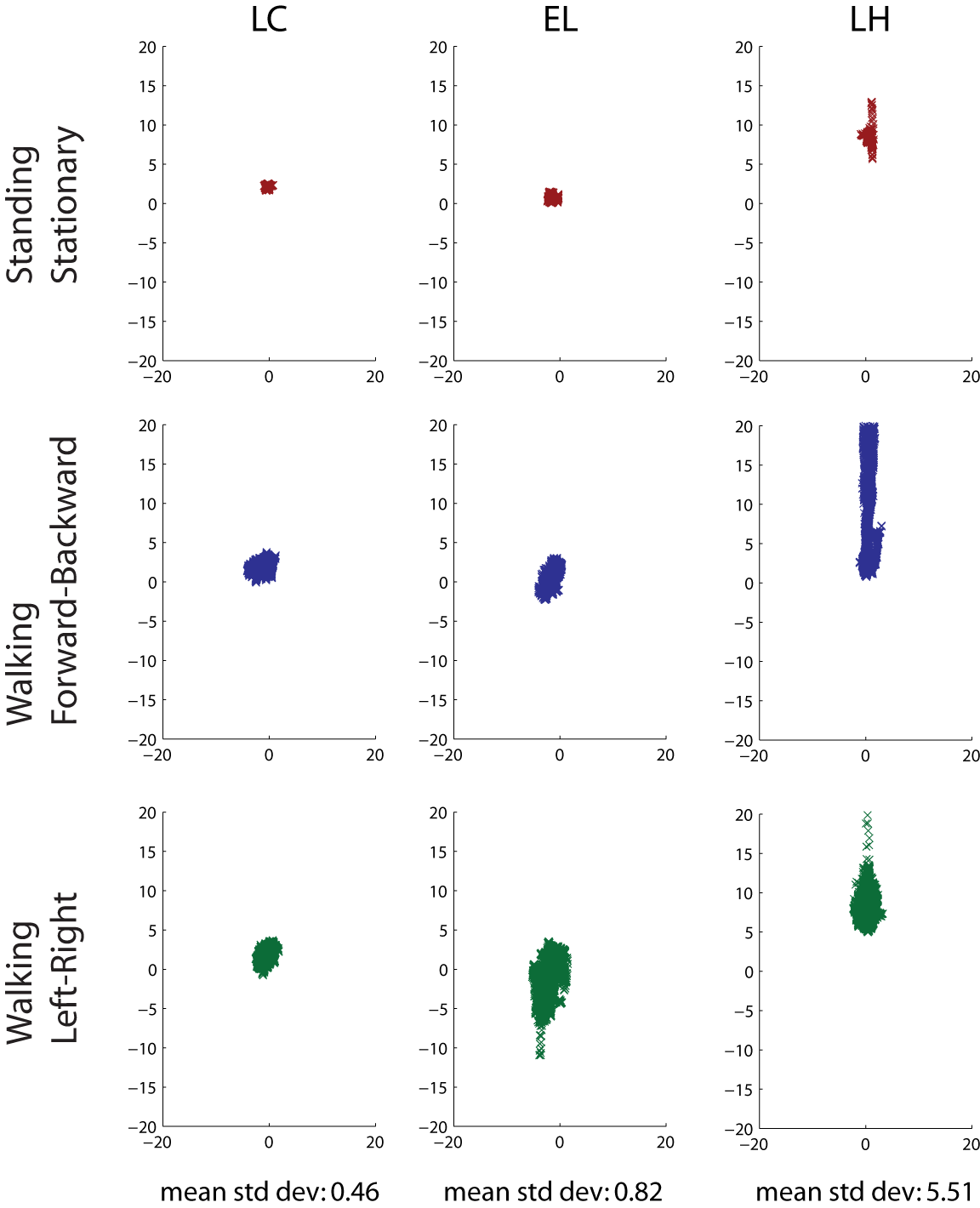
The experimental software for performing these evaluations are available for download ([www.sourceforge.net/projects/libgaze](http://www.sourceforge.net/projects/libgaze)).

Robustness to body movement

Our system should provide consistent gaze measurements even when the user is mobile. To test this, we instructed twelve participants to maintain gaze fixation at a single fixed point, which was displayed at eye-height on the screen, while performing various body movements. They were instructed to either stand perfectly still, strafe horizontally (left-right), walk towards and away from the screen (forward-backward). When standing still, participants were approximately 100cm away from the screen. When required to move, participants first took a single step to the left (or forward), returned to the starting position, a step to the right (or backward) and finally, back to the starting position. Each of these movements were performed once, after the participants were fully calibrated while sitting down.

For each body movement, we calculated the angular difference between the *true* and *measured* gaze vector for every recorded estimate (approximately 120Hz), after blinks were removed. The *true* gaze vector was derived by calculating the vector that described the spatial relationship between the displayed stimulus and the current origin of gaze in the WCS. This vector represented perfect fixation. The current gaze vector on the other hand was the actual gaze of the participant, as estimated by our system. This angular difference was decomposed into its vertical and horizontal component and a normal distribution was fitted to the measurements obtained in trial. Figure 5 shows the average offset of estimated gaze from the ideal gaze vector, across the three types of body movement and its range. These errors and variance in gaze estimations are expected to have resulted from both the participants’ errors in fixation and instability of the recording headset.

The extent to which body movement disturbs gaze estimates in our system, vary across different participants. In Figure 6, the gaze trace of the last trial of each body movement is illustrated for the best, median and worst participant.

Table 1 summarizes the variance of an average user’s gaze on each trial, in terms of the gaze’s horizontal component, vertical component and their combination. The largest variance during movement is in the vertical component of the user’s gaze. In addtion, forward-backward movement introduces the most overallvariance in gaze measurements, although this varies across individuals This data is taken to indicate that drifts occur most often during forward-backward walking and in a way that particularly affects the vertical component of our gaze data.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Horizontal | Vertical | Combined |
| Standing Still | 0.30  (0.01) | 0.57 0.41 | 0.53°  (0.41) |
| Forward-Backward | 1.10  (0.13) | 2.01 (10.32) | 2.03° (10.65) |
| Left-Right | 1.09  (0.13) | 1.21  (0.59) | 1.06° (0.32) |

Table Average variance (individual variance in brackets) of the user’s gaze in terms of its horizontal, vertical and combined components, when fixating a single stimuli while standing still, walking left-right or forward-backward.

Calibration errors across display

A fundamental assumption in our computations of gaze is that the standard calibration, which is performed on only the central sub-region of the display and when the head is in a neutral straight-ahead orientation, can be generalized to the rest of the display. In other words, we assume that our system calibration is valid regardless of head orientation. This is unlikely to be true, as head and body movements are expected to displace the headgear. which will introduce errors to our measurements. In the first evaluation, we directly compared errors in the validation of our system calibration, across different head orientations.

Fifteen participants took part in this evaluation and were required to perform a standard calibration procedure (see Implementation) on each trial. There was one variationn, however, and it was that the validation phase was now conducted twice; once with the head oriented in neutral position, as during calibration, and the second time, with the head oriented towards one of seventeen possible points. The calibration was performed on a series of fixation points taken from a 3 x 3 grid, with the dimensions of 40° x 40° in visual angles. The first validation of the calibration was also performed with the head in a straight-ahead orientation, on fixation points which were drawn from a 3 x 3 grid, with the reduced dimensions of 30° x 30°. If the mean and maximum validation errors did not exceed 2.5° and 3.0° respectively, a second validation was performed with the head oriented towards a new screen position. Each trial took about 1 min to complete.

Figure . Gaze traces of the three participants are rank-ordered for the gaze variance (left to right).

A large blue dot was presented prior to each calibration and validation procedure to guide participants to each new head-to-screen orientation. The participants’ current head orientation to the screen was continuously displayed by a red dot, which had to be positioned within the blue dot before either the calibration or validation procedure was conducted. Seventeen different head positions were sampled for the second validation and these were evenly spaced points, in steps of 10°, along the cardinal and their intermediate axes.

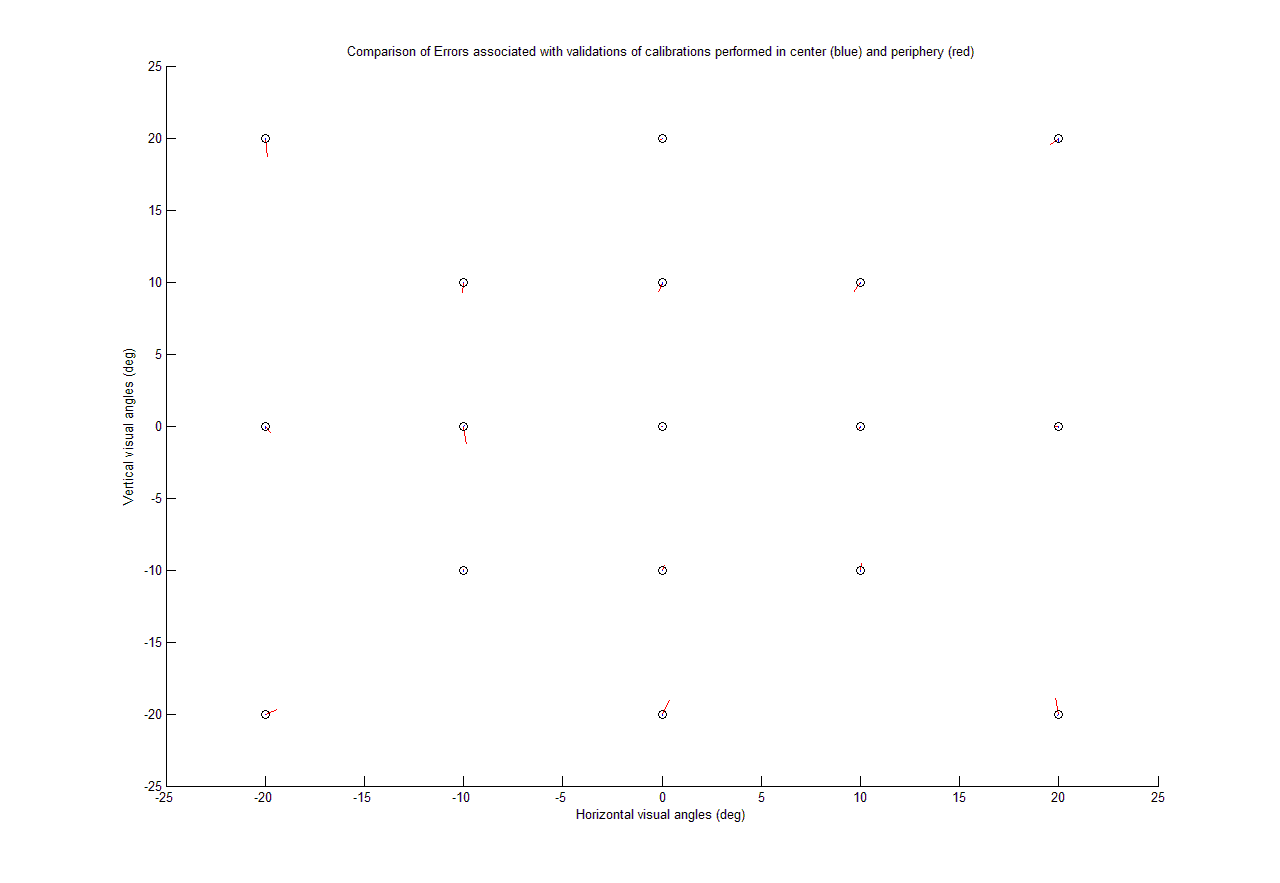


Figure : Comparison of the mean validation errors associated with a head-eccentric position (blue) versus a head-centric position (red). Each line indicates the direction and magnitude of the validation error.

Figure . The single-step gaze shifts of two participants, from a central fixation cross to an off-center visual probe (30 deg to the right), involves an eye and head movement. *Include a plot of other people’s data for comparison.*

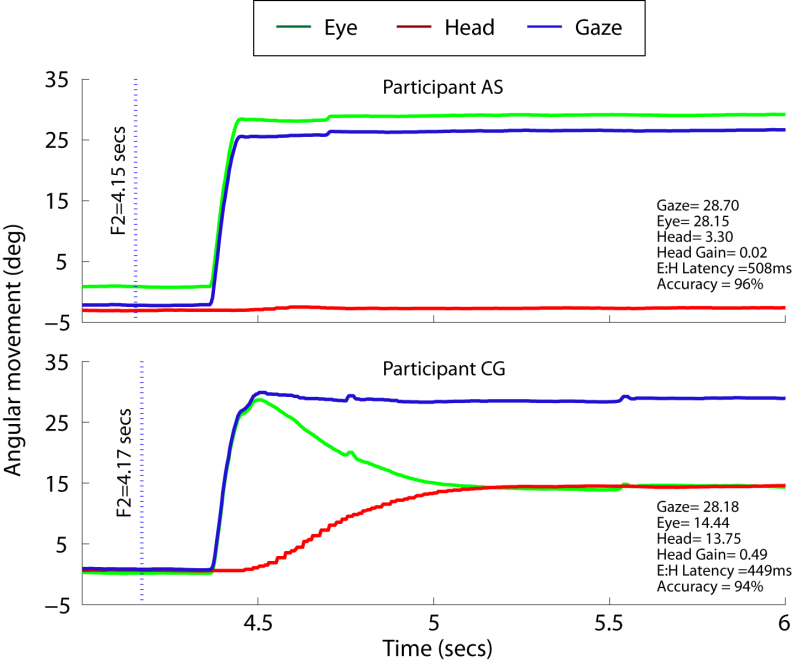
A mean drift error was derived for each validation procedure, for each eye. Therefore, each participant produced seventeen drift errors for the head neutral validation, as well as its accompanying head off-center validation. Four data points exceeded 5° and occured simultaneously in both eyes. These treated as blinks and were excluded from from the analysis.

The standard calibration procedure, which is performed and validated with a neutral head position, yields a mean validation error of 0.70 (standard deviation = 0.22). In contrast, the validation of the same calibration produces a mean error of 1.16 (standard deviation = 0.63) if performed with an off-center head orientation. Individual participant data are plotted in Figure ??. From this, it is seen that there is an increase in error when the head is not in a neutral position, for at least 8 participants.

Typical eyetracking experiments conducted on natural scene viewing (e.g., ) often accept calibration with 0.5° to 0.75° of validation error.

When validation of the calibration procedure is performed with a change in head alignment, a systematic increase drift error occurs. The degree to which this occurs is individual dependent and is likely to result from mechanical slippage of the headgear. Inertia causes the head gear to be biased towards the center and hence, gaze measurements are shifted towards the middle of the display. There is a calculated gain of ?? error, for every vertical angular difference in head alignment to the center and ?? error for horizontal angular differences. These errors should be taken into account when analyzing data.

Accuracy in natural gaze measurements

Figure 7 illustrates how gaze movement is a combination of eye and head movement. The magnitude of each component is known to vary across individuals (Fuller, 1999) and we show the same here. We demonstrate the data obtained from our system is qualitatively comparable to the existing scleral search coil method.

Finally, we measured the variance in our estimated fixations , resulting from saccades of varying directions. In each trial, participants first presented a fixation dot on the screen that they had to orient their heads towards as well as fixate. After 1000ms, this fixation dot disappeared and a second dot appeared in the center of the screen, which they were required to shift their gaze towards. When performing this gaze shift, participants were instructed to either move their eyes only, or to reorient both their eyes and head to the new fixation cross in the screen center.

There were eight possible starting positions drawn from the border of a 3 x 3 grid. Each position was used four times, which resulted in a total of 32 trials; that is, 6 x 4 cardinal directions and 4 x 4 diagonal directions. Each position was separated from its nearest cardinal neighbor by 20° and from its diagonal neighbor by 28.3°.

This evaluation was conducted twice for each participant. One set of trials required participants to maintain their original head orientation (*head-moved*) during gaze shift and the other required participants to reorient their gaze and head to the new position (*head-unmoved*).

The findings from this evaluation is summarized in . The mean error for *head-moved* and *head-unmoved* were 1.23° and 1.88° respectively. A paired t-test reveals that the *head-unmoved* condition results in overall errors that are (marginally) significantly larger than the *head-moved* condition (t(6)=2.09, p<0.08). There was no noticeable trend in the accumulation of error across trials for either conditions.

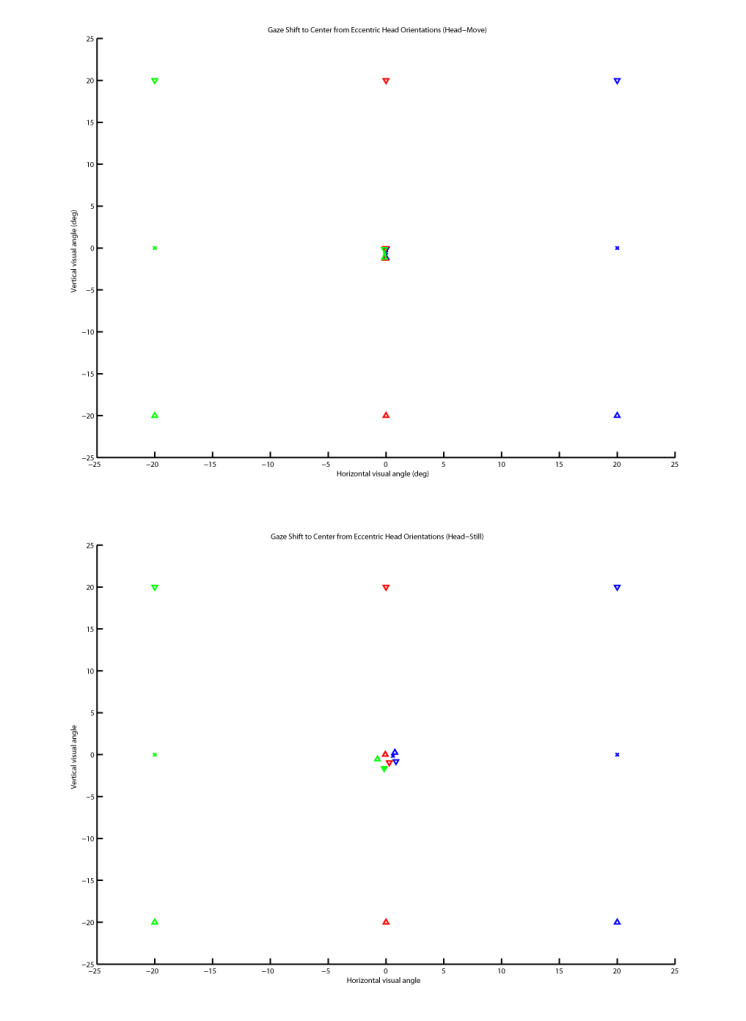


Figure 9. Screen positions of starting head orientations and their associated mean errors when reorienting to the center.

|  |  |  |
| --- | --- | --- |
| Saccade Direction | Head  unmoved | Head  moved |
| (5) 20° ↑ | 1.59 (0.85)  [-0.06, 0.01] | 1.48 (0.68)  [-0.06, -1.24] |
| (2) 20°→ | 1.37 (0.54)  [0.60, -0.15] | 1.09 (0.43)  [0.01, -0.53] |
| (4) 20° ↓ | 1.83 (0.74)  [0.29, -0.95] | 0.94 (0.34)  [-0.04, -0.05] |
| (7) 20° ← | 2.09 (1.27)  [-0.11, -1.55] | 1.22 (0.53)  [-0.11, -0.75] |
| (3) 28.2° NE | 1.76 (0.75)  [0.77, 0.27] | 1.58 (0.67)  [0.01, -1.17] |
| (1) 28.2° SE | 1.88 (0.80)  [0.87, -0.80] | 1.00 (0.39)  [0.04, -0.08] |
| (6) 28.2° SW | 2.54 (1.68)  [-0.16, -1.63] | 1.00 (0.31)  [-0.14, -0.14] |
| (8) 28.2° NW | 2.00 (0.79)  [-0.74, -0.53] | 1.49 (0.49)  [-0.13, -1.14] |
| Overall | 1.88  [0.18, -0.66] | 1.23  [-0.05, -0.64] |

Table . Mean angular distance between final gaze and target stimulus. Standard deviations are reported in brackets. Mean horizontal and vertical offsets of final gaze, relative to the target, are reported in the square brackets.

Natural gaze behavior

Here, we present gaze data that were collected using our system, from participants who performed typical eyemovement task. Specifically, gaze data from participants who were required to make gaze shifts, to freely view natural images and to visually search natural images for prespecified content.

Discussion

Error variance in tracked data tend to result from slippage in the head markers, used to denote gaze origin in WCS. This varies, depending on how well the device fit to each individual’s head shape. Such errors can be corrected by using goggle-based eyetrackers that ensure a tighter fit (Babcock & Pelz, 2004). In addition, markers can be placed closer to the eyes. This will help in reducing the magnitude of moment that arises from small movements.

Here, we show that the accuracy of our measurements are limited by:

1. eyetracker slippage
2. calibration techniques
3. virtual model of the environment

Is it important to allow for unrestrained head movements during gazetracking? Head movements have largely been ignored in gazetracking research, because models of gaze control have suggested that the same saccadic eye movement is programmed regardless of head movement; that head movements which occur during a saccade simply serve to attenuate the saccade, by action of the vestibulo-ocular reflex (VOR), by the amount equal to head displacement (referred to as ‘VOR-saccade interaction’) (Bizzi et al., 1971). This oculo-centric view of eye-head coordination has been popular with vision researchers for its simplicity.

When the head is immobilized, activity in the superior colliculus (SC) is associated with eye saccades with a specific direction and amplitude (Robinson 1972; Schiller and Stryker, 1972). In reality, however, SC neurons exhibit activity that is related to the combined eye-head movements rather than to either the eye or head component alone (Freedman and Sparks, 1997). This implies that the standard depiction of the SC motor map, as obtained in head-restrained setups, is a systematic underestimation of the amplitudes of gaze movements.

Our understanding of eye-head coordination in gaze control has progressed through the systematic work of Zangemeister, Stark, e.t.c. However, little of this has influenced of visual cognition

[[discuss difference in calibration methods from Jeffreys02 and Ronsse07. Ronsse07 calculated his gain matrix by perform a best fit by linear regression. In contrast, we employ Moore96’s video based algorithm with fixed threshold settings for error tolerance. Ronnse07 calibration is easier and faster to conduct (i.e., 20s)]]

both = assume that the orientation of the head-tracked object is equal to the natural eye viewing direction (so 0.0, 0.0 deg)

one = tracks the eye position with markers on the face -use head realted viewing vector for fittng mapping function

the other one = -calculates eye position by assuming headpos = eye pos. Then they do a validation on a different position to use the error to assume a correct eye position (asumption error comes from wrong eye position)

The Calibration-Validation evaluation suggest that head movements are accompanied by an approximate increase in error of up to 0.46°. This error varies across individuals and results from movement of the head-mounted eyetracker. At a distance of 100cm to the current display screen, this approximates to 0.8cm (or 5 pixels). While the current system can still be improved by reducing slippage of the eyetracker, it is adequate for most psychophysical experiments. Another aspect is the need for participants to maintain a fixed head position in a non-rest head posture. Individuals could vary in their ability to do so because of differences in muscular neck strength.

Display modules: A configuration for the planar display module, consisting of four points and representing a display model, can be created by a pyGaze tool. The configuration of the created planar display model is set so that [0,0]=top left, [1280,1034]=bottom right.

To create planar display configuration. Planar display coordinates: [0,0]=top left, [1280,1034]=bottom right. This is determined by VisionEggVisualHooks because typically, VisionEgg uses OpenGL

The influence that minor variations in the experimental setup designs can yield over behavioral performance should not be underestimated. For example, by increasing the cost of information acquisition from a simple saccade to a head-movement, Ballard induced a shift from a memoryless strategy to one that required holding information in working memory (Ballard, Hayhoe, & Pelz, 1995; Ballard, hayhoe, Pook, & Rao, 1997). Conversely, allowing head-movements during gaze shifts and the resulting increase in the number of small-amplitude saccades might result in a different cognitive strategy from when head is restrained.

Appendix A

If there is more than a single appendix, name them Appendix A, Appendix B, etc.

Acknowledgments

This research was supported by a grant from the Baden-Würrtemberg xxx (BW-FIT) as part of the “Information at Your Fingertips” partnership. We wish to thank the following researchers for their help testing the system with a wide variety of hardware and application scenarios: Werner König, Joachim Bieg, Harald Reiterrer, Oliver Deussen, xxx, Berlin group.

Commercial relationships: none.

Corresponding author:

Email:

Address: Spemannstr. 38, 72076, Tübingen, Germany.

References

Hyperlink each in-text citation to its reference anchor. Insert the [PubMed] link as well using the example given here (see [instructions.pdf](http://journalofvision.org/info/instructions.pdf)). If the full text is available from an archival on-line source (e.g. Journal of Vision), that should also be provided as a link to the string [Article] at the end of the reference.

Andrews, B. W., & Pollen, D. A. (1979). Relationship between spatial frequency selectivity and receptive field profile of simple cells. *Journal of Physiology,* *287*, 163-176. [[PubMed](http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&list_uids=430391&dopt=Abstract)]

Babcock, J.S. & Pelz, J.B. (2004). Building a lightweight eyetracking headgear. ETRA: Eye Tracking Research and Applications Symposium, 109-113.

Bizzi, E.; Kalil, R. E. & Tagliasco, V.  
Eye-Head Coordination in Monkeys: Evidence for Centrally Patterned Organization.  
*Science,* **1971***, 173*, 452-454

Chen, J., Solinger, A. B., Poncet, J. F. & Lantz, C. A., 1999, Meta-analysis of normative cervical motion.  
*Spine, 24*, 1571-1578

Guitton, D. & Volle, M. (1987). Gaze control in humans: eye-head coordination during orienting movements to targets within and beyond the oculomotor range.  
*Journal of Neurophysiology, 58*, 427-459

Ronsse, R., White, O., & Lefevre, P. (2007).   
Computation of gaze orientation under unrestrained head movements. *Journal Of Neuroscience Methods, 159*, 158-169.

Moore, S. T., Haslwanter, T., Curthoys, I. S. & Smith, S. T.  
(1996). A geometric basis for measurement of three-dimensional eye position using image processing.  
*Vision Research*, *36*, 445-459.

Peirce, JW (2007) PsychoPy - Psychophysics software in Python. J Neurosci Methods, 162(1-2):8-13

Peirce, JW (2009) Generating stimuli for neuroscience using PsychoPy. Frontiers of Neuroinformatics, (2008) 2:10. doi:10.3389/neuro.11.010.2008

Zangemeister, W. H. & Stark, L.  
Types of gaze movement: variable interactions of eye and head movements.  
Exp Neurol, 1982, 77, 563-577