Eye-Tracker Validity Documentation

Introduction:

The purpose of this document is to explain to a user how the eye tracker's measurements were tested for validity. It will discuss the methodology that we used to collect sample points from the eye tracking device, as well as the way that we compared the data that we obtained with the data that is output by the proprietary software.

Methodology:

The motivation for our experiment is to collect gaze data from a subject, but we would also like to ensure that the data that is collected is valid data. The simplest way to do this is to create a point of fixation at a known point on the screen, and then have the subject fix their gaze on that point, while the eye tracker is collecting the data. The easiest known point on the screen to do this with is the center of the screen, since it is easy to find out the dimensions—in pixels—of the screen, and calculate the center from those dimensions. Ideally, the subject's gaze would be locked on the center of the screen, such that the eye tracker would constantly record the same position. However, this is not the case, as our eyes make very minute movements that we do not notice, and it may be very difficult to keep them completely centered on the point of fixation.

Data collection from the eye tracker can be done in two ways. The first way to extract the data from the eye tracker is three functions in the code that will be displaying a stimulus. These functions are from the API for the eye tracking device. The three functions are named iV_StartRecording, iV_StopRecording, and iV_SaveData. To collect data using these functions, iV_StartRecording must be called prior to displaying any stimuli, and iV_StopRecording must

be called when the stimuli is finished being displayed. After this, the data must be saved as an IDF via a call to iV_SaveData. This file is an Intermediate Data File (IDF), and as its name implies, it is not the complete data file; it must undergo another process before we can actually obtain the actual data.

In order to obtain the data, the data in the IDF must be exported to a .txt format by the BeGaze software. To do this, store the IDF and stimuli in a folder, and open the BeGaze software, and go to "File => New Experiment from Folder." Once there, select the directory in which the IDF file was placed in. If you have the IDF file in another computer, transfer it to the computer with the BeGaze software, and place it in a directory, along with the stimuli that was used during the experiment. Once this has been done, hit "Next" in the window that appears. There will be a beeswarm video on the screen that can be played. This video shows a path of where the user's gaze was throughout the data collection process. This video will also be cross-referenced with an image of all of the (x,y) coordinate points collected, by plotting the coordinate points on top of the image, and by visual inspection, determining if the beeswarm video and the image with the points plotted onto it match.

To decode the data in the IDF, we go to "Export => Export Raw Data to File," and select the beeswarm<timestamp>.avi file on the left. When determining what to export, select Pupil Diamter[mm] from Raw Data, and Gaze Position from Points of Regard (POR), Also, ensure that the only channel that we will receive data for is the left eye, so uncheck the right eye if it has a checkmark. Finally, proceed to the Details tab, and ensure that the "Write Header" option is not checked.****NOTE: THIS IS VERY IMPORTANT, AS THE SCRIPT THAT WAS WRITTEN TO COMPARE THE TWO DATA FILES REQUIRES THAT THE BEGAZE DATA FILE***

NOT HAVE A HEADER. THIS IS WHAT IS KNOWN AS A CONDENSED FILE* Also, ensure that the decimal places box is set to 3, and that the separator box is set to Comma. Finally, hit export, and the file will be a .txt file in the same directory as where the IDF is located. Open the file, and resave it as a CSV file.

The second method of data collection does not rely on the eye tracker's software, but purely on the functions in the API. The process is similar to the first process described, except that in the parts of the code where the stimuli is being shown, we must query the eye tracker to get the information about its latest sample. To do this, we use the function iV_GetSample from their API, and use a SampleStruct—for more information about this struct and all other structs, please refer to pages 27-33 of the iView X™ SDK 3.0 manual—to store the information about both eyes. Throughout the time that the stimuli are displayed, we constantly keep querying the data from the eye tracker, using calls to iV_GetSample, and we record the data in vectors. When the stimuli is finished being shown, the data in the vectors can be written out to a CSV file, and analyzed. The data obtained is the same as that obtained from the BeGaze software.

As a result of these observations, we have decided to assess the validity of the data collected by the eye tracker by designing our experiment as follows:

Procedure:

1. Use a small cross whose center is placed at the center of the screen. This will be the stimuli.
2. Design a script that will begin by calibrating the eye tracker, and not allow the experimenter to move on, until all visual angles of deviation are below 0.5˚--visual angle deviations are returned in degrees--as mentioned in page 74 of the Red-m eye tracking system manual. Also, the script should record eye tracking data using the two aforementioned data collection

methods.(This step has already been done. To take a look at the code, please refer to testEyeTrackerValidity.m). Also, note that calibration deviation parameters should also be recorded within a CSV file.

3. Once the data has been collected, plot a left eye gaze x position vs. time graph, and an eye gaze y position vs. time graph.

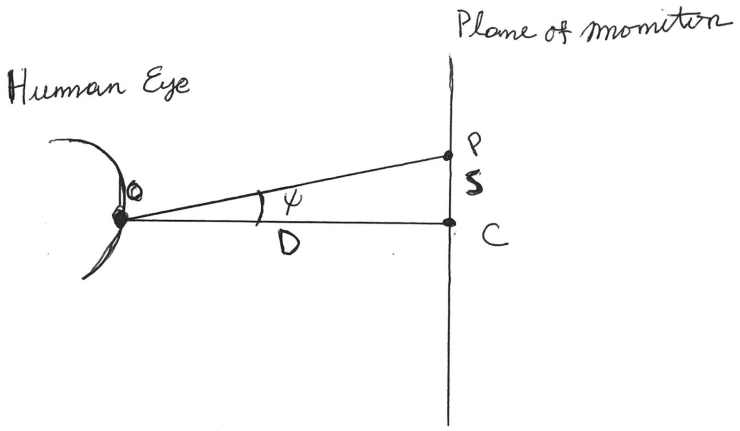
4. On the x position vs. time graph, also plot a line whose value is the horizontal center of the screen.

5. On the y position vs. time graph, also plot a line whose value is the vertical center of the screen.

6. Use BeGaze to extract the left eye gaze x and y positions, as well as the eye diameter.

Analysis:

Once the data has been collected from both BeGaze and our script, we will perform a series of tests based off of the data that we have obtained. First, it is important that the data in our CSV file matches the data obtained by BeGaze, so ensure that the data matches in these files by running the matchData.m script. Once that test has been passed, observe the x position vs. time and y position vs. time graph. Ideally, most of the data points should be in close proximity to the horizontal and vertical centers of the screens. To calculate the tolerance by which they can be off center, we must first delve into the meaning of the visual angle deviation parameters that we obtained.



When we look at the image of the cross, the image subtends an angle*, Ψ--CAUTION: Ψ is returned in degrees, so make sure that it is converted to radians*--to our eye, as seen on the diagram. This angle that it subtends is the angle of deviation that is reported to us by the calibration parameters. This angle, along with the distance *D*, as seen on the diagram, can be used to obtain the how off center the user's eye can be, by allowing us to calculate *S*, the length at which the user may be off by, and then converting that to pixels. The mathematics behinds it is seen here:

From the image, we can derive that:

$$tan(\Psi) = \frac{S}{D} \Leftrightarrow S = Dtan(\Psi)$$

And to convert to pixels, we let the height of the screen in pixels be *P* and its actual height in cm be *H,* and it follows then that the number of pixels per cm:

$$\frac{P}{H} \Rightarrow S(Pixels) = \frac{PDtan(\Psi)}{H}$$

And if we let *Y* be the midpoint of the screen, since that is where the center of the cross is, then it follows the possible tolerance values that the data points should be within are:

$$Y \pm \frac{PDtan(\Psi)}{H}$$

Now that we know this, we should ensure that most of the data points fall within the calculated boundaries.

Once this has been done, the final test that will be conducted is to test that the data points in the data that we collected using the data that we obtained mirrors the data bee swarm path that can be obtained using the BeGaze software. To do this, we must write a script that will plot the points on the image of the cross that we are using. This has already been done. Once all three of these tests have been passed, we can determine that the eye tracker is properly collecting data.

SAMPLE CALCULATION

Data Sheet

* Add to doc.

ypositive Tolerance: 542.4788 84
yneg Tolerance: 507.525 116
Arclength = 507. 525 488.
yMP = 525

Using formula: $\frac{PD\tan(\psi)}{H} = L$

P = 1080 pixels
H = 29.61 cm
D ≈ 60 cm.
ψ = 0.4575°

$$\frac{(1080)(60)(\tan(0.4575))}{29.61} = \text{error}$$

17.47488372 pixels = error

yMP ± error = ypos tolerance, yneg Tolerance.

Arclength = Radius * ψ

$$= 60\text{cm} * \left(0.4575\left(\frac{\pi}{180}\right)\right)\left(\frac{1080\text{pixels}}{29.61\text{cm}}\right)$$

$$= \boxed{17.47451233\ \text{pixels}}$$

yMP ± Arclength = 507. 525 4877. pixels.