Using Unity to Obtain Eye-Tracking Data From the VIVE Pro Eye Headset

Zachary Hoffman

Department of Human Physiology, Research Mentors: Dr. Andrew Karduna, Kate Spitzley

Introduction

The growing virtual reality (VR) market has driven the research and development of new technological features, such as the addition of eye-tracking cameras, to some VR models.

Figure 1. *The VIVE Pro Eye VR Headset used (left), and an example of the cameras within the headset tracking an eye.*

From the location of the pupil, the headset can gather where the user is looking, called the gaze direction.

VR allows researchers to manipulate the visual and auditory sensory information a participant in a virtual environment is receiving.

The output and analysis of gaze data can be applied to further biomechanics research, and to other fields with research related to the analysis of subject gaze.

The aims of this project were to create a virtual test environment in Unity with a moving object, and to have an eye-tracking code that would compare the gaze of the subject to the position of a moving visual target at each frame.

Virtual Test Environment Creation

Within the room was a pink target coordinate plane that the moving target object would traverse across.

Upon putting on the HMD, the subject would appear to be standing in the center of this room, (underneath a spotlight).

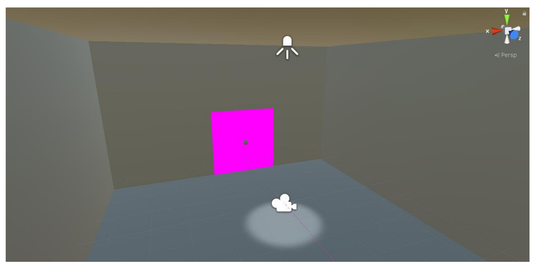

Figure 2. *The virtual test environment with a camera icon to indicate the source location of the subject's field of view.*

Software Development Kit

Before any coding, the VIVE Pro Eye software development kit (SDK) SRanipal was installed and imported into Unity.

With modifications to a gaze visualization script (GazeRaySample) in the SDK, gaze data could be acquired and exported from Unity.

Variable Initialization

Within the GazeRaySample script, variables were created to acquire, store, and modify the gaze data coming from the eye-tracking cameras.

A variable GazeDirectionCombined was created to combine the direction each eye was looking into one intermediate vector.

```
Vector3 GazeDirectionCombined = Camera.main.transform.TransformDirection(GazeDirectionCombinedLocal);
GazeRayRenderer.SetPosition(0, Camera.main.transform.position - Camera.main.transform.up * 0.05f);
GazeRayRenderer.SetPosition(1, Camera.main.transform.position + GazeDirectionCombined * LengthOfRay);
```

Figure 3. *The code used to acquire the combined gaze direction from the two eye-tracking cameras.*

GazeDirectionCombined allowed for acquisition of the x,y,z components of the Gaze direction Vector3 data coming from the eye-tracking cameras.

Next, a variable was created to store these individual x,y,z components of the gaze direction Vector3, and a time counter, each frame.

A second variable was also updated each frame to compare the position where a subject's gaze converged with the position of the moving ball at each frame.

These variables were necessary to store the information the eye-tracking cameras were receiving, so that it could be compared to the position of the visual target.

```
void Shoot ()
{
    RaycastHit hit;
    if (Physics.Raycast(Camera.main.transform.position, GazeDirectionCombined, out hit))
    {
        compare_variable += hit.transform.position;
        compare_variable += ", ";
        compare_variable += movingball.transform.position;
        compare_variable += ", delta: ";
        compare_variable += (movingball.transform.position - hit.transform.position);
        delta_look = (movingball.transform.position - hit.transform.position);

    }
}
```

Figure 4. *The Raycast function used to acquire positional data for the surface a subject's gaze was on.*

The RaycastHit function works like firing a bullet in a video game; it only outputs data each frame if the subject gaze ray hit a surface.

Variable Update Order

```
// Updated each frame to record the Vector3 for the combination of each gaze vector
eye_look = GazeDirectionCombined.ToString();
gaze_x = GazeDirectionCombined.x;
look_variable += time_counter;
compare_variable += time_counter;
compare_variable += ", ";
look_variable += ", ";
look_variable += gaze_x;
look_variable += ", ";
gaze_y = GazeDirectionCombined.y;
look_variable += gaze_y;
look_variable += ", ";
gaze_z = GazeDirectionCombined.z;
look_variable += gaze_z;
Shoot();
compare_variable += "\r\n";
look_variable += "\r\n";
time_counter += Time.deltaTime;
```

Figure 5. *Code demonstrating how each variable was updated and stored within look_variable to be formatted correctly in the subsequent text file.*

At each frame, a time counter was added into the variables storing the gaze data.

These variables were then updated to additionally hold the gaze position Vector3 of the subject, and the individual x, y and z components of their gaze direction.

This chain of code formatted the data in a way that allowed it to be easily analyzed by other software when exported as a text file.

Coding Instructions for the Visual Target

The moving target was given parameters for the minimum and maximum positions it could oscillate between.

Another variable was created to specify whether movement would be exclusively along the x or y axes.

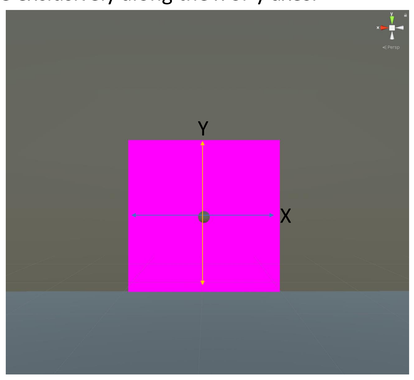

Figure 6. *Subject's perspective of the moving ball, and the plane behind it, with arrows added to demonstrate the two planes it can possibly move between.*

Results

Subject gaze data was acquired (n=1) from the eye-tracking cameras, along with comparison data for the difference between the moving ball's location and the gaze position of the subject.

```
0.300882, (-1.7, 1.2, -2.1), (0.0, 1.4, -4.2), delta: (1.7, 0.2, -2.1)
0.3201943, (-1.7, 1.2, -2.1), (0.0, 1.4, -4.2), delta: (1.7, 0.2, -2.1)
0.3324186, (-1.7, 1.2, -2.1), (0.0, 1.5, -4.2), delta: (1.7, 0.2, -2.1)
0.3441845, (-1.7, 1.2, -2.1), (0.0, 1.5, -4.2), delta: (1.7, 0.2, -2.1)
0.3564817, (-1.7, 1.2, -2.1), (0.0, 1.5, -4.2), delta: (1.7, 0.3, -2.1)
0.3719149, (-1.7, 1.2, -2.1), (0.0, 1.5, -4.2), delta: (1.7, 0.3, -2.1)
0.3826393, (-1.7, 1.2, -2.1), (0.0, 1.6, -4.2), delta: (1.7, 0.3, -2.1)
0.3934784, (-1.7, 1.2, -2.1), (0.0, 1.6, -4.2), delta: (1.7, 0.3, -2.1)
0.4047947, (-1.7, 1.2, -2.1), (0.0, 1.6, -4.2), delta: (1.7, 0.4, -2.1)
0.4166041, (-1.7, 1.2, -2.1), (0.0, 1.6, -4.2), delta: (1.7, 0.4, -2.1)
0.4271336, (-1.7, 1.2, -2.1), (0.0, 1.7, -4.2), delta: (1.7, 0.4, -2.1)
0.4402389, (-1.7, 1.2, -2.1), (0.0, 1.7, -4.2), delta: (1.7, 0.4, -2.1)
0.4526355, (-1.7, 1.2, -2.1), (0.0, 1.7, -4.2), delta: (1.7, 0.5, -2.1)
0.4752771, (-1.7, 1.2, -2.1), (0.0, 1.8, -4.2), delta: (1.7, 0.5, -2.1)
```

Figure 7. *Example text data output from the code, organized as: time, gaze position, ball position, and difference between the two (listed as a Vector3).*

Conclusions

The aims of this project were to create a virtual test environment in Unity with a moving object, and to have an eye-tracking code that would compare the gaze of the subject to the position of the moving object at each frame.

The aims of this project were met, and all data was successfully exported in a format that can be analyzed in another software (i.e Microsoft Excel).

This programming framework could be used to acquire data for eye-tracking research in Biomechanics and other fields.

Without the restrictions of the COVID-19 outbreak, I would have liked to use this framework to conduct research evaluating the oculomotor system with Fitt's Law.

Acknowledgements

I would like to thank my research mentor Kate Spitzley and lab principal investigator Andrew Karduna for this research opportunity.

References

1. Tieri, Gaetano, et al. "Virtual reality in cognitive and motor rehabilitation: facts, fiction and fallacies." Expert review of medical devices 15.2 (2018): 107-117.
2. Mirelman, Anat, et al. "Effects of virtual reality training on gait biomechanics of individuals post-stroke." Gait & posture 31.4 (2010): 433-437.
3. Subramanian, Sandeep, et al. "Virtual reality environments for post-stroke arm rehabilitation." Journal of neuroengineering and rehabilitation 4.1 (2007): 20.
4. "VIVE Pro Eye Specs: VIVE™." VIVE, www.vive.com/eu/product/vive-pro-eye/specs/.
5. Iskander, Julie, Mohammed Hossny, and Saeid Nahavandi. "Biomechanical analysis of eye movement in virtual environments: A validation study." 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2018.
6. Prostheticknowl. "Prostheticknowledge." Prosthetic Knowledge, 10 July 2018, prostheticknowledge.tumblr.com/post/175747127281/eye-tracking-in-the-htc-vive-pro-short-video-from