Getting started with Unity and VR

Version 1.0

September 13, 2022

AdHawk Microsystems

410 Albert St, unit 102, Waterloo, ON, N2L 3V3, Canada

info@adhawkmicrosystems.com

Contents

Getting started with AdHawk eye tracking and the Meta Quest 2 VR headset

What is eye tracking?

Eye tracking is the process of observing eye behavior so you can learn about the point of gaze. Eye tracking can be used to:

- learn what someone is looking at
- learn about aspects of someone's mental state (e.g., fatigue, concussion, level of focus, etc.)

Human eye movements

Before diving into the AdHawk SDK and writing an application that incorporates eye tracking data, it helps to know about some of the more common eye movements.

Fixations

One of the most common eye events occurs when the eyes aren't moving. *Fixations* occur when we focus on something for a period of time. Fixations range in duration from tens of milliseconds to several seconds.

Saccades

Saccades occur when our eyes move between fixations. These movements are the fastest movements the human body can perform; they can reach a peak velocity of 500 degrees/second (although the amplitude of saccades are usually in the 4 to 20 degree range).

Smooth pursuit

Smooth pursuit is another form of eye movement but one which requires something to follow (unlike saccades which can happen without stimuli). Consider a bird flying across the sky: our brain enters a feedback loop where we can smoothly track the bird as it moves. Smooth pursuit velocities range from 10 to 30 degrees/second (compared with the much faster saccades).

Blinks

Blinking closes the eyelids to moisten the eye. We often think of blinking as not involving eyeball movement, but while fixating on something straight ahead, the eyeball actually rotates downward slightly during a blink.

> Blinks are an event within the AdHawk API; you can register to receive notifications when they occur.

Vestibulo-ocular reflex (VOR)

The vestibulo-ocular reflex acts to stablize your gaze (on a visual target) while your head is moving. This helps stabilize the image on your retina as the environment may be changing. To see VOR in action, fixate on a word in this sentence and move your head from right to left; notice how your gaze remains locked to the word while your head moves?

One of the calibration methods supported by AdHawk relies on VOR ('fixed gaze').

Vergence

Vergence involves eyes moving in opposite directions. This is done to converge or diverge, usually to focus on items that are close up or far away, respectively.

Vergence information is available as part of the gaze data stream.

How does eye tracking work?

You can generally tell where someone is looking by looking at their eyes. Eye tracking is nothing more than a really accurate, really fast way of the same thing.

Some eye tracking systems use a combination of small cameras pointed at your eyes and image analysis to figure out where you're looking.

AdHawk eye tracking doesn't use cameras and image processing (both require a fair bit of power), instead using incredibly small (MEMS-scale) mirrors to shine a lower power IR light across your eyes. Reflections from your eyes are captured by photodetectors and feed into a mathematical model. This model allows us to determine where you're looking very accurately (< 1 degree MAE) and at high rates (up to 500 Hz).

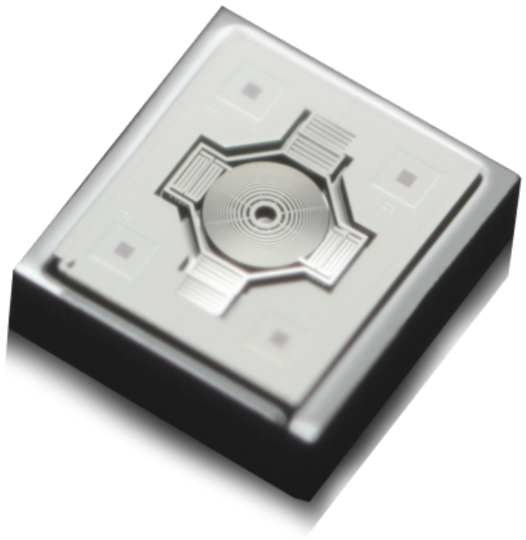

Figure 1: MEMS mirror

Eye tracking applications

Eye tracking can be used in several different ways:

- **Interaction**. In the same way as a mouse or your finger can move across a screen, so to can your gaze act as a pointer. Consider actions like blinks to be like mouse clicks or taps. This sort of direct interaction can be as rich as other modalities, and can allow us to communicate our interest and intent through eye behavior.
- **Observation**. Eye behavior is becoming increasingly valuable in unlocking mental and cognitive states. Eye tracking can be used to observe and collect information about these states in an unobtrusive way. Data collected can be used to help identify, diagnose, or assess recovery for certain medical conditions (e.g., concussion, dementia, etc.).

- **Research**. Although a specialized form of observation, eye tracking has been used in market research (to learn more about people's behavior in retail environments, for example) and lab settings (e.g., psychology, neuroscience, usability, etc.) for a long time.

We've posted a few interesting examples of eye tracking on our Hack the North website.

AdHawk eye tracking

Traditional eye tracking systems point a camera at your eye and use the image of your pupil to determine where you're looking.

The AdHawk eye tracking system does not use a camera to calculate your gaze. Instead, a path of low power IR light is drawn on your eyeball and photodetectors capture the reflections. Those glints tell enough about the position and orientation of your eyes to figure out where you're looking.

This system has been inserted into the Meta Quest 2 headset to provide high quality eye tracking within a VR environment. With this approach we can calculate your gaze (where you're looking), the vergence angle (angle between your individual eye gaze vectors), and pupil size and position.

Eye tracking data is made available to your application through several Unity components and scripts.

System requirements

To create an eye tracking application that runs in VR on the Meta Quest 2 headset, you'll need a reasonably powerful, Windows-based computer capable of driving the headset display over a link cable (provided as part of the kit).

On the computer, you'll use Unity to develop your application, then run it on the Meta Quest 2 headset using the link cable. The computer must:

- run Windows 10 or Windows 11
- have 2 USB-A ports (one for the link cable, one for the eye tracker)
  - the Oculus Link port on the computer side needs to be a USB 3.0 port
- have a discrete GPU (see Meta's list of supported GPUs)
- have Unity 2021.1.21f1 installed (that's the version we use; others may work) with:
  - Windows Build Support
- Visual Studio Community 2019 (or other Unity-compatible debugging tool/text editor)
  - this can be installed through Unity Hub

Many newer laptops do not have two dedicated USB ports or a discrete GPU suitable for this work, so a recent gaming laptop may be required.

You must also have a smartphone with the Meta Quest (Oculus) mobile app (for iOS 10+; Android 5.0+) installed. A Meta account is also required but can be created within the mobile app. The Meta Quest 2 headset will connect to your smartphone using Bluetooth, and use the credentials you've created or signed in with on the mobile app.

For more information about system requirements of the Meta Quest 2, see the Oculus Link PC system requirements.

Hardware components

The hardware components within the AdHawk eye tracking system are:

1. **Eye tracking inserts**. The eye tracking components (scanner, photodetectors) are contained within modules that are placed over and around the Meta Quest 2 headset lens mounts.

- a small PCB sits within the headset and manages the eye tracking system and data streams
- a USB cable connecting the eye tracking system PCB and your computer

1. **Meta Quest 2 headset**. The eye tracking system works in tandem with the headset.

- a USB cable connecting the headset and your computer

Software components

To create a VR application with eye tracking, you will use:

- Unity (and a few dependencies: *Visual Studio Community* and *Windows Build Support*; see above)
- the AdHawk Unity SDK (available on AdHawk's Hack the North site)

Installing the SDK

The AdHawk Unity SDK is packaged as a .zip file:

- unzip the package to a directory on your computer
- open the project from within Unity Hub

We have provided Unity projects which incorporate key parts of the AdHawk Unity SDK. These projects demonstrate how to:

- set up and calibrate eye tracking within a VR app
- set up your VR scenes within Unity
- receive, process, and act on eye tracking data
- interact with scene elements using gaze

Working with Unity

Ensure that you have a Unity Personal license created and installed. A personal license can be added through Unity Hub (Profile > Manage licenses > Add).

Fitting the Meta Quest 2 headset

Eye tracking systems are largely comprised of physical sensors positioned near the eyes themselves. To ensure great eye tracking, you need to optimize the position of those sensors.

For a good fit using the Meta Quest 2 headset:

- **Remove your glasses**. While very narrow glasses do fit within the Meta Quest 2 headset, prescription glasses prevent eye tracking. For more common prescriptions, some lenses are available from the AdHawk sponsor booth.
- **Adjust the IPD setting for the headset**. The interpupillary distance (IPD) is the distance between your pupils when you're looking into the distance. The Meta Quest 2 headset supports three IPD values: 1 (58 mm), 2 (63 mm) and 3 (68 mm). Meta describes how to adjust this on their support site.
- **Tighten the straps**. To reduce slipping due to the weight of the headset, you should adjust the straps so that the headset is snug but not tight. General instructions for fitting the headset are provided by Meta on their support site.
- **Consider using the provided spacer**. The Meta Quest 2 kit includes a 'glasses spacer' which can be stacked between the headset and stock facial interface. Most people don't require this, but if other adjustments don't help, consider trying it.

Calibration

The process of calibration allows the system to relate a measured gaze to the virtual environment. That is, if you're looking up and to the left, we want to make sure that the gaze information we provide your application is correct.

In general, **you should run a calibration whenever someone puts on the headset**. You should also support someone explicitly running a calibration from within your application if it makes sense (e.g., where progress or work would be lost if they had to restart in order to calibrate). You can do this by revisiting the calibration scene within your Unity app.

The example projects demonstrate best practices for your VR application by having a calibration scene after the app intro scene. This allows users to get set up for the best eye tracking possible. During development you may want to perform a re-center; this is a quick way to recenter your gaze within a scene. Re-centering isn't as comprehensive as a full calibration, but can do in a pinch during development. TODO: Add reference to re-centering keyboard shortcut.

Creating a VR app with eye tracking

The Meta Quest 2 headset you've been provided includes an AdHawk eye tracking system. With this hardware and by using the specialized eye tracking components and scripts we've provided as part of the Unity sample project, you can:

- calibrate the eye tracker
- use gaze information in real-time within your Unity application
- receive notifications about events such as blinks to help with interaction

Ideal VR app flow

When creating an eye tracking app in VR, there are a few steps that need to occur in a specific order. Each of these steps are represented as scenes in your Unity project.

1. **Welcome the user**. It makes sense to introduce the user to the environment, especially when it requires some specialized setup like calibrating the eye tracker. This scene orients the user to your application and how to use it.
2. **Perform a calibration**. As a rule of thumb, each time someone puts on the headset, you should run a calibration. Moving the headset too much—such as taking it off and putting it on—will change the position of the eye tracker relative to the eyes. When this happens, a calibration will fix things up.
3. **Your application**. This is where you shine. But keep in mind that there should likely be a way to explicitly trigger a calibration within the application in case eye tracking isn't great. By allowing the user to calibration from any point in your application, you allow them to retain their progress.

Designing your VR scenes

Field of view

Eye tracking with the Meta Quest 2 is best within a 40 degree (horizontal) by 25 degree (vertical) area. While eye tracking is possible outside of this zone, there may be less accuracy. "Eye-interactive content"—content that can be interacted with using eye movements—within a VR environment should be placed within this ideal eye tracking zone.

Interactions involving eye tracking should be located centrally to keep things comfortable.

Positioning and sizing interactive elements

There are a few different scenarios to consider when positioning and sizing eye-interactive elements.

Interactive elements that are in a fixed position relative to the user (e.g., a heads up display (HUD)) should be placed 1 m from the user viewpoint. That distance has been shown to be comfortable with respect to vergence (how your eyes move to accommodate for depth).

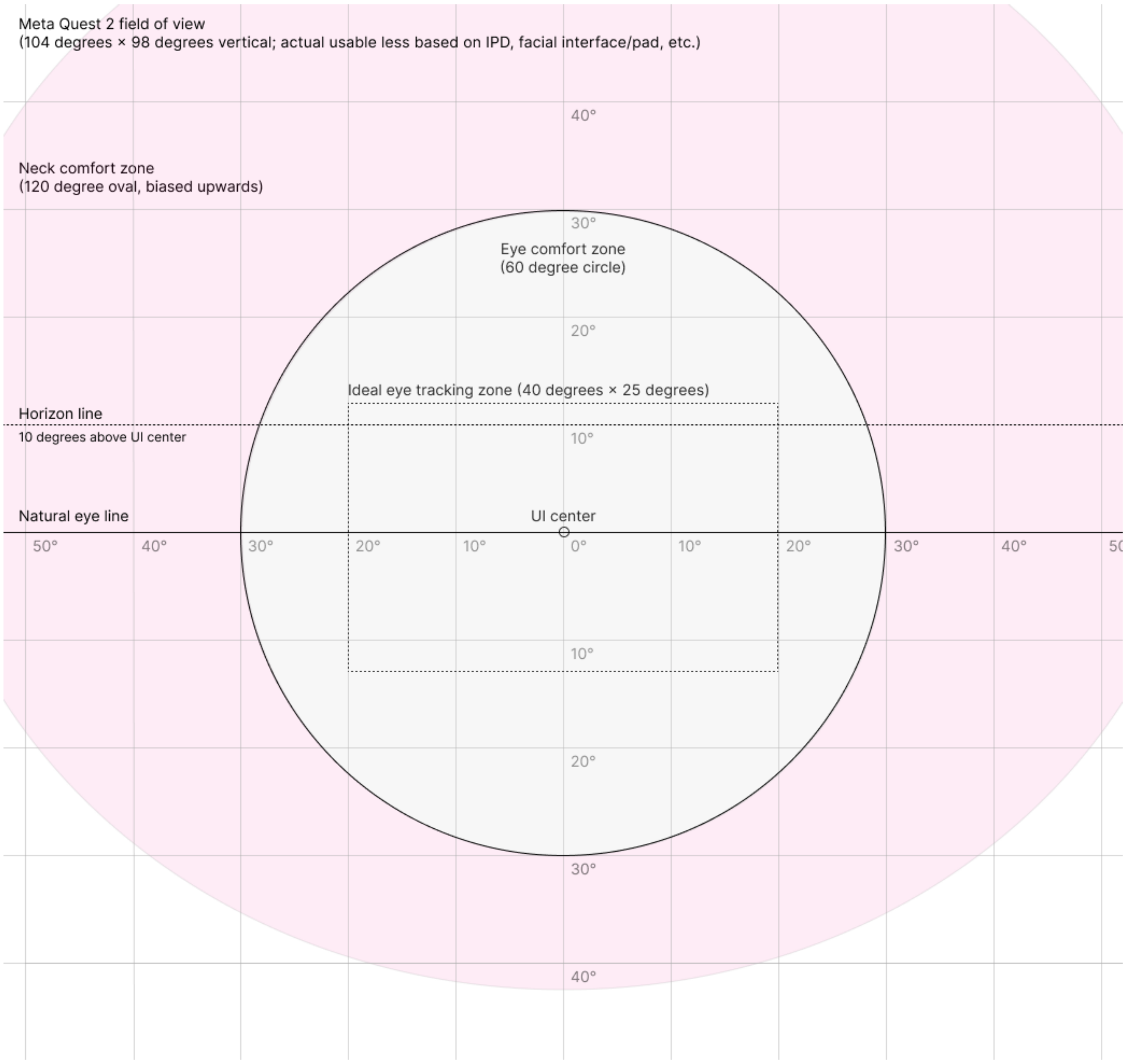
Meta Quest 2 field of view
(104 degrees × 98 degrees vertical; actual usable less based on IPD, facial interface/pad, etc.)
Neck comfort zone
(120 degree oval, biased upwards)
Eye comfort zone
(60 degree circle)
Ideal eye tracking zone (40 degrees × 25 degrees)
Horizon line
10 degrees above UI center
Natural eye line
UI center
40°
30°
20°
10°
0°
10°
20°
30°
40°
50°

Figure 2: Fields of view in VR

If your VR application supports moving around the scene and interactive elements are part of that scene, think about the size of those interactive elements. If users will likely be far away, consider making the elements (or their 'hit boxes') larger and attention-grabbing.

Important Unity components and scripts

We've provided components to help you get eye tracking working quickly. Lean on these components

GazeEventSystem

The `GazeEventSystem` component allows gaze events to be received as mouse events. This means that:

- eye movements will be translated into mouse-like movements for 2D canvas objects in world space
- blinks will be translated into mouse clicks
- allows interaction with 3D colliders that have the GazeDetector component

GazeDetector

The `GazeDetector` component is an event system compatible interaction handler. It allows you make objects that are aware of gaze and gaze events (such as `OnGazeEnter`, `OnGazeLeave`, and blink events).

EyeTrackerAPI

The `EyeTrackerAPI` component is a singleton that allows for access to gaze data, events streams, and other eye tracking-related functions. This will help you use AdHawk eye tracking within your application, and acts as a bridge between the lower level eye tracking API and Unity.

ControllerInputHandler and KeyboardInputHandler

The `ControllerInputHandler` and `KeyboardInputHandler` components simplify working with the Meta Quest 2 controllers. In some situations you may need to use the physical controllers, but keep in mind that gaze and associated eye events (such as blinks) are a great way to interact as well.

Learn from the example scenes

Each of the example scenes provided follow the recommended flow of: welcoming the user, performing a calibration, and then presenting the main application scene. They also show how to use the components we've provided to help get robust eye tracking into your app.

Basic eye tracking scene ("3 Your App Here")

This scene contains all of the base components that you'll need to get started with eye tracking. When building your application, start from here and build out.

Interacting with objects ("BalloonPopping")

This scene contains a few 'balloons' and demonstrates the gaze detector in 3D space. If you look at the balloon it will expand. If you look away it will stop growing. If you blink, the balloon you're looking at will pop.

This code has a `GazeEventSystem` and uses `GazeDetector`.

Saccade selection ("SaccadeSelectExample")

This scene demonstrates the `GazeEventSystem` and `GazeDetector` components. In this scene, you can look at one object, drag it down—with your gaze—to another object as a way to 'apply' it. This demonstrates multi-point interaction.

This code has a `GazeEventSystem` and uses `GazeDetector`.

Button pressing ("ButtonPress2DWorldspace")

This example shows off `GazeEventSystem` and `GazeDetector` in a 2D canvas. In this scene, if you look at one button, another button appears. Looking at the top button swaps the appearance of that button while you look at it. If you blink while looking at the top button, the bottom buttons reappear.

An important note: There is a lot of power in the Unity event system when used with the Unity Inspector and its ability to drag and drop components into event system slots. This *drag and drop 'coding'* gets used a lot and is very powerful for prototyping within Unity.

This example will show you how to have an object react to a blink.

Testing gaze with a mouse during development

The AdHawk Unity SDK includes several components to route eye tracking data to your application. However, sometimes it is useful to be able to run your application on your computer (not on the VR headset); in these situations, you can configure a mouse to simulate gaze.

- within the Unity Inspector, configure the eye tracking source (`ET Source`) for `EyeTrackerAPI`
- you can choose to use a mouse or the backend (the headset's eye tracking source)
- when you build the VR version of the application, `ET Source` will be explicitly set to Backend (which uses the AdHawk eye tracking hardware within the headset)

Key steps

There are a few steps required to get eye tracking working well within your application:

1. Ensure that AdHawk Backend is running on your computer. It will be available in the System Tray.
2. The headset must be plugged in:

- both the Oculus VR headset and the eye tracking insert inside the viewport must be connected via the provided USB cables to your computer

1. The Oculus app should be running on the development computer.
2. Enable Oculus Link (Meta has a video to explain how to do this).

Working with data

Eye tracking data is provided through `EyeTrackerAPI`. This is a singleton object that you can use within your Unity applications to access:

- EyeTrackerAPI.Instance.Streams.Gaze
- EyeTrackerAPI.Instance.Streams.Events

Gaze data

Gaze data is accessed via `EyeTrackerAPI.Instance.Streams.Gaze.Position`.

Event data

Blink events are likely the most useful when developing interactive gaze-enabled VR applications. Such events are represented by `EyeTrackerAPI.Instance.Events.Blink`.

`EyeTrackerAPI.Instance.DidBlinkLastFrame` can be quite useful when working with blink events. It is a boolean and is true if the user finished a blink in the last frame.

`GazeEventSystem` has a tool which will provide a pointerClickInteraction at the position that the gaze was at just before the blink started. This can be useful in processing a blink event (e.g., to determine what the user is trying to interact with).

When handling blink events, it is important to note that blinks aren't instantaneous: there is a start time, a middle, and an end time. Between the start and end time, the gaze vector for that eye will move around a bit in unpredictable ways (this is because the eye moves when eyelids close). Users should take this into account when using `DidBlinkLastFrame`—you may want to ignore the gaze vector during the blink itself and perhaps for a short period of time thereafter.

Troubleshooting

Gaze vector is not accurate after running a calibration

If the gaze vector doesn't appear to be tracking your actual gaze accurately, even after running a calibration:

- check your headset fit (see the fitting section of this guide)
- re-run calibration

Gaze vector is not accurate after headset slips

When the position of the headset changes significantly relative to your eyes, eye tracking can suffer. Check your headset fit (see the fitting section of this guide) as a starting point, in particular:

- tighten the headset so that it doesn't shift or droop
- try the spacer on the headset (or remove it if you've already been using it)

Oculus Link button does not show up

If the Oculus Link button is no longer available within the Oculus Quick Settings:

- unplug the headset side of the Oculus Link cable
- firmly replug in the Oculus Link cable on the headset side

Headset stays gray or black when worn

When the headset stays gray or black after you've put it on, it isn't recognizing that you're using it. You can:

- ensure the proximity sensor (within the headset) is not blocked
- restart the headset (hold power button until it turns off; turn it back on again)
- ensure the battery is charged (battery life is about 2–3 hours of use; this will be longer if use isn't constant or the headset is plugged into a PC)

Problems with scene orientation

When you start your application, are you positioned away from where you expected? Are you looking in the wrong direction? If you encounter these issues:

- look in the right direction and position yourself properly
- start Oculus Link
- start your app after Oculus Link is running

Objects don't react to gaze events

If you're finding that an object in your scene isn't responsive to gaze events:

- ensure there isn't another collider in front of your object
- check that the object uses GazeDetector

Glossary

AdHawk MindLink. AdHawk MindLink glasses include a full eye tracking system within the frames.

Anti-saccade. A saccade *away* from a stimulus.

ArUco markers. Specialized (graphical) markers designed to facilitate computer vision. They are blocky black and white grids with a black border.

Auto-tune. A process to optimize where the eye tracking system will scan on someone's eyes.

Calibration. The process by which the relationship between a user's eye tracking setup—and thus their gaze—and the world is calculated, allowing for the 'real world' gaze target to be determined.

Device calibration. Infrequently, he eye tracker needs to be calibrated against a known environment or set of inputs. This is different than regular calibration (see above) and involves a calibration *fixture*. If you need to perform a device calibration (e.g., you get a return code of 17 when calling into the API), visit AdHawk's sponsor booth.

Eye tracking. The process of inferring the gaze point (and, in general, eye behavior) by using technology to observe someone's eyes.

Fixation. Fixations are when we focus on something for a period of time. Fixations range in duration from tens of milliseconds to several seconds.

IPD. Interpupillary distance. The distance between the centres of your eyes/pupils (whilst looking straight ahead, far into the distance). This value is often part of your eyeglass prescription.

MAE. Mean absolute error. Used to characterize the quality of a calibration (the higher the MAE, the worse the calibration). MAE is calculated as part of a validation. The acceptable range of MAE depends on the type of calibration performed: the more points used during the calibration, the higher the expectations for its quality (and thus a lower MAE). For a 1 point calibration, the largest MAE 'accepted' is 2°; for a 9 point calibration, a 1° MAE is the upper limit.

Saccade. The rapid movement of both eyes to shift the centre of gaze to a new portion of the visual field.

Scene. Within Unity, a scene contains the objects for your game or application. A single game or application can have multiple scenes; tying them together allows you to build a story or change contexts.

Tracker. The collective term for the hardware and software used to track a single eye.

Quick Start. A process to get the eye tracker set up quickly. You can trigger a Quick Start via the API.

Validation. The process of having a user look at a series of targets post-calibration. By comparing the calculated gaze to known target position, the validity of the calibration can be assessed.

VOR. Vestibulo–ocular reflex. Reflex triggered by the vestibular system (related to the inner ear and orientation sensing) to ensure that the visual signal to the eye is stabilized during movement. One of the calibration modes ('VOR') we use involves keeping your gaze fixed on a point and moving your head to align a 'cursor' to that fixed marker; this takes advantage of the VOR. See https://en.wikipedia.org/wiki/Vestibulo%E2%80%93ocular_reflex