Unity Data Collection for Meta Quest Pro

Contents

[Copyright 1](#_Toc563397243)

[Getting Started 1](#_Toc1330567871)

[Software 1](#_Toc371280953)

[Hardware 1](#_Toc821843114)

[Running the Project 2](#_Toc1197538398)

[Project Overview 2](#_Toc651189003)

[Objects in the Scene 2](#_Toc15738113)

[GazeRecorder 2](#_Toc913887691)

[WorldReferenceFrame 2](#_Toc1769844127)

[CameraReferenceFrame 2](#_Toc1233488498)

[HeadReferenceFrame 2](#_Toc260293061)

[CyclopeanEyeReferenceFrame 2](#_Toc436727503)

[LeftEyeReferenceFrame and RightEyeReferenceFrame 2](#_Toc1149475399)

[OculusInteractionSampleRig 2](#_Toc561692339)

[OculusInteractionSamplesRayCanvas (1) 2](#_Toc1152614228)

[EventSystem 2](#_Toc122736079)

[TaskController 2](#_Toc804531600)

[ValidationResult 3](#_Toc723397679)

[How Is Gaze Data Being Recorded? 3](#_Toc572726741)

[Task Sequence 3](#_Toc1976026262)

# Copyright

Copyright 2024, Texas State University. All rights reserved.

This work is licensed under a Attribution-NonCommercial-ShareAlike 4.0 International. See license conditions at:<https://creativecommons.org/licenses/by-nc-sa/4.0/deed.en>

For all other uses, please contact the Office for Commercialization and

Industry Relations at Texas State University <http://www.txstate.edu/ocir/>

# Getting Started

## Software

The following software was used for our development of the project:

* Windows 10
* Visual Studio Community 2019
* Unity
  + Unity Hub 3.4.1
  + Unity Editor 2021.3.17f1
  + Packages
    - Meta Movement 1.3.3
    - Oculus Integration 47.0
    - Oculus XR Plugin 3.2.2
    - OpenXR Plugin 1.6.0
    - TextMeshPro 3.0.6
* Meta Quest
  + Meta Quest Developer Hub (MQDH) 3.2.0
  + Oculus App 49.0.0.170.358

## Hardware

The following equipment was used for our development of the project:

* Desktop computer
* Meta Quest Pro headset
* USB-3 to USB-C cable (for using Quest Link during development)

# Running the Project

Open the project in Unity (takes a couple of minutes). The “MeasureSignalQuality” scene should be opened by default. With the Meta Quest Pro connected to the computer via Quest Link, press the Play button in Unity. The camera should follow the headset’s orientation. UI elements can be interacted with via a “pinching” hand gesture.

To run the project without Quest Link, go to File > Build Settings, and click Build. This will create a .apk file which can be installed to the Meta Quest Pro headset via MQDH.

# Project Overview

## Objects in the Scene

### GazeRecorder

Responsible for querying the latest gaze information and writing recordings to output files.

### WorldReferenceFrame

Making this the parent of an object will position that object relative to the world. Currently, this is simply an identity transformation; but in case that ever needs to change, this reference frame can be used.

### CameraReferenceFrame

Making this the parent of an object will position that object relative to the camera. It seems that this is currently equivalent to the HeadReferenceFrame, but this is provided in case that is ever not true.

### HeadReferenceFrame

Making this the parent of an object will position that object relative to the head. It seems that this is currently equivalent to the CameraReferenceFrame, but this is provided in case that is ever not true.

### CyclopeanEyeReferenceFrame

Making this the parent of an object will position that object relative to the cyclopean eye. Currently, for the Meta Quest Pro, the cyclopean eye is the average of the left and right eyes.

### LeftEyeReferenceFrame and RightEyeReferenceFrame

Making either of these the parent of an object will position that object relative to the respective eye.

### OculusInteractionSampleRig

This is provided by an Oculus package to drive interaction with the Meta Quest Pro. Particularly of note is that we use the OVRCameraRig’s eye tracking capabilities and the InputOVR’s hand tracking/interaction capabilities.

### OculusInteractionSamplesRayCanvas (1)

This is an interface designed to be interacted with via “pinching” hand gestures.

### EventSystem

Necessary for handling interactions with the interface.

### TaskController

Manages virtually everything in the scene. Responsible for instantiating eye tracking tasks, hiding/showing the interface, acting on button presses, and controlling when gaze data is saved, among other things.

### ValidationResult

A canvas for displaying spatial accuracy during the validation task.

## How Is Gaze Data Being Recorded?

Gaze is recorded in the “RecordGaze” script (attached to the “GazeRecorder” object in the scene) which is responsible for querying the latest gaze information and writing recordings to output files. The latest gaze sample is queried every frame using OVRPlugin.GetEyeGazesState(). If the approach to querying gaze position ever changes, all relevant changes should be contained within the RecordGaze.GetGazePosition() method. After a task finishes, data is written to a file in the directory specified by Application.persistentDataPath. Recorded files can be found in the directory “Android/data/com.OKLab.EyeTrackingDemo/files” in the headset’s storage.

## Task Sequence

1. RAN127 (Random saccades with lighter gray background)
2. BROW (Raising/lowering eyebrows)
3. WINK (Closing/opening eyes)
4. RAN63 (Random saccades with darker gray background)
5. VOR (Rotate head while looking at world-locked stimulus)

Calibration and headset fit optimization are each performed once before beginning the program. Each task is preceded by a short, 5-point validation procedure to check spatial accuracy during the experiment. Validation data can also be used for offline calibration later.