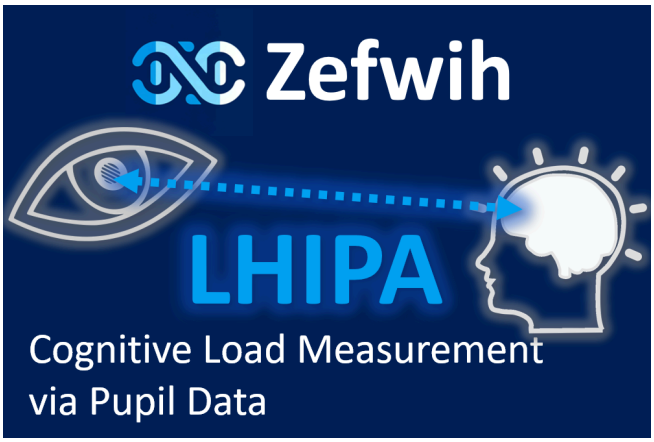
LHIPA - Low-High Index of Pupillary Activity and Mental Load Calculator



Unity Asset Documentation

v1.0

Introduction

The **LHIPA (Low-High Index of Pupillary Activity)** is a cognitive workload metric based on fluctuations in pupil diameter. It is widely used in cognitive science, human-computer interaction, and psychology to assess mental effort in real-time or post-processed data analysis.

What is LHIPA?

LHIPA is computed by analyzing high-frequency changes in pupil diameter using **Symlet16-Wavelet transformation**. It is independent of sampling rate and provides a standardized measure of cognitive load.

A **higher LHIPA value** indicates increased cognitive demand, while a **lower value** suggests minimal effort or distraction.

- **Example Values:**
  - 0.05 = Low cognitive load
  - 0.25 = High cognitive load

Scientific Background

For more details on LHIPA, refer to:

Duchowski, A. T., et al. "**The Low/High Index of Pupillary Activity (LHIPA): A Psychophysiological Measure of Cognitive Workload**". CHI Conference on Human Factors in Computing Systems, 2020.

Asset Key Features

✅ **Plug & Play:** Drag the script into your Unity scene and call the LHIPA function with pupil diameter data—computation happens automatically!
✅ **Cross-platform:** Works on **desktop, mobile, and VR** applications.
✅ **Live & recorded data:** Supports **real-time LHIPA computation** and post-analysis of **pre-recorded eye-tracking data**.
✅ **Configurable settings**
✅ **Well-documented & example scenes:** Includes **two demo scenes** showcasing LHIPA integration.

Technical Details

How LHIPA is Calculated

LHIPA is computed by analyzing the **power of high-frequency bands** in the pupil signal using a **Symlet16-Wavelet transformation**. This method detects changes in pupil size associated with mental workload.

What's Included in the Asset?

- **LHIPA Calculation Script** – The core script for computing LHIPA.
  2 Example Scenes:
  - **Eye Tracking Simulator:** Simulates real-time LHIPA computation.
  - **LHIPA Testing Scene:** Allows testing with recorded pupil diameter data.
- **2 Prefabs:**
  - **Eye Tracking Simulator UI** – Example UI for live data processing.
  - **LHIPA Tester UI** – UI for analyzing recorded data.
- **Example pupil diameter data** in `pupil_diameter_signals.json`.

Technical Requirements

- Unity **2020.1 or newer**
- **TextMeshPro** (required for UI elements)
- **Newtonsoft.Json** required for reading and writing recorded json data.

You can install both of the packages in the Unity Package Manager (*Window -> Package Manager -> TextMesh Pro* and *Window -> Package Manager -> Newtonsoft.Json for Unity*)

Usage

LHIPA.cs is the main script and *CalculateLHIPA()* the main method you need to calculate a LHIPA value. Just call *CalculateLHIPA()* with an array of pupil diameters and provide the recording time in seconds and get the LHIPA value. The method can be called **in real-time** (with a running eye tracker) or **retrospectively** (using recorded data).

CalculateLHIPA() Method Inputs

- float[] pupilDat: Array of pupil diameters in mm (min. length: 256)
- float durationInSeconds: Time between first and last data point
- float modMaxCorrectionThreshold: Threshold for modulus maxima correction
- bool debugLog: Enable detailed logging in Unity Console

CalculateLHIPA() Method Output

- float LHIPA: Value between 0 and 1 representing cognitive load

---

Scripts & Methods

1. LHIPA.cs

Description

This is the **core script** for calculating the **Low-to-High Index of Pupillary Activity (LHIPA)** based on a **pupil diameter signal**. (See above)

modMaxCorrectionThreshold Details

This threshold is used to **eliminate noise** in modulus maxima values that arise from rounding errors.

Best correction values:

- `0.01 - 0.1` recommended
- Higher values may reduce noise but affect sensitivity

Example regression results:

Threshold	R^2	F-Value	β	α
0.0	0.31	43.08	0.14	0.043
0.05	0.056	5.86	0.096	0.065
0.1	0.049	5.06	0.097	0.065

---

2. LHIPATester.cs

Description

A testing class to compute LHIPA values for **recorded data or simulated input arrays**. Further examples, how to use *CalculateLHIPA()* are in the Start()-Method.

Public Variables

- public float durationInSeconds: Duration of the data sample
- public float modMaxCorrectionThreshold: Correction threshold for LHIPA
- public string inputFilePath: Path to recorded eye-tracking data
- public string outputFilePath: Path for LHIPA result logs
- public bool consoleLog: Enable logging in Unity console
- public float[] eyeDiametersExampleArray: Example pupil diameter data

Public Methods

TestLHIPAAndLog(bool useInputFile)

Computes LHIPA for test data and logs results.

Parameter:
useInputFile: If true, uses recorded data; if false, generates test arrays

Returns:
public string: Path to logfile with LHIPA values

GenerateTestArrays()

Generates 100 sample pupil diameter arrays for testing.

Returns:
List<float[]>: List of float arrays</returns>

---

3. EyeTrackSimulation.cs

Description

Simulates eye-tracking data to test LHIPA calculations in Unity. *CalculateLHIPA()* is called live every *calculationInterval* seconds. Can be used as an example, how to work with a real eye tracker.

Public Variables

- float baselineDiameter: Average pupil diameter
- float simulationSwitchInterval: Time between high/low pupil sizes
- float samplingRate: Simulated eye tracker sampling rate
- bool startWithScene: Auto-start simulation
- float calculationInterval: LHIPA calculation interval
- float modMaxCorrectionThreshold: Correction threshold
- bool showDetailedCalculationLog: Enable debug logging
- float currentLHIPAValue: Stores latest LHIPA value

Public Methods

StartSimulation()

Starts eye-tracking simulation and the live calculation.

StopSimulation()

Stops eye-tracking simulation and the live calculation.

GetPupilDiameterFromEyeTracker()

Returns:
float: Random pupil diameter

Additional Components

UI Scripts

Handle example **UI interactions** with canvas (button clicks, text field inputs, displaying the results) in the provided prefabs and scenes.

SharpWave (Wavelet Library)

Modified sharpWave implementation with Symlet16 and updated decompose()-methods. Based on sharpWave by graetz23: https://github.com/graetz23/sharpWave under MIT License: https://github.com/graetz23/sharpWave/blob/master/LICENSE

Can be replaced with other wavelet implementations if needed.

Contact

For any queries or issues related to this module, or if you are interested in scientific collaboration, please contact us at:

contact@zefwih.com

www.zefwih.com