



HAL
open science

ArGaze: An Open and Flexible Software Library for Gaze Analysis and Interaction

Théo De La Hogue, Damien Mouratille, Mickaël Causse, Jean-Paul Imbert

► **To cite this version:**

Théo De La Hogue, Damien Mouratille, Mickaël Causse, Jean-Paul Imbert. ArGaze: An Open and Flexible Software Library for Gaze Analysis and Interaction. 2024. hal-04564025v1

HAL Id: hal-04564025

<https://enac.hal.science/hal-04564025v1>

Preprint submitted on 30 Apr 2024 (v1), last revised 12 Sep 2024 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

**ArGaze: An Open and Flexible Software Library for Gaze Analysis and
Interaction**

Théo de la Hogue¹, Damien Mouratille¹, Mickaël Causse², and Jean-Paul Imbert ¹

¹École Nationale d'Aviation Civile, Toulouse, France

²ISAE-SUPAERO, Toulouse, France

Author Note

Correspondence concerning this article should be addressed to Jean-Paul Imbert,
ENAC 7 avenue Edouard Belin CS 54005 31055 Toulouse Cedex 4,
jean-paul.imbert@enac.fr.

Abstract

Gaze analysis has evolved into a mature technique with diverse applications enabling the investigation of various human activities and cognitive processes such as reading, visual attention, memory as well as real time gaze interaction. However, existing proprietary software or libraries often lacks the flexibility needed to incorporate emerging gaze metrics or real time processing, limiting researchers to predefined methodologies.

To address these limitations, we introduce ARGAZE, an open and flexible software library designed to provide a unified and modular approach to gaze analysis or gaze interaction. ARGAZE facilitates real-time and/or post-processing analysis for both screen-based and head-mounted eye tracking systems. By offering a wide array of gaze metrics and supporting easy extension to incorporate additional metrics, ARGAZE empowers researchers and practitioners to explore novel analytical approaches efficiently.

This paper first reviews existing software solutions and their functionalities before delving into the design principles and features of the ARGAZE library. Subsequently, three use cases demonstrate ARGAZE's efficacy in processing eye data from screen-based and head-mounted eye trackers, showcasing its versatility and applicability across different research and practical contexts.

Keywords: eye tracking, gaze analysis, software, library, Python, real-time, modular

**ArGaze: An Open and Flexible Software Library for Gaze Analysis and
Interaction**

Introduction

Gaze analysis is now a mature technique used in a wide range of applications and research. It enables us to study an impressive range of human activities and cognitive processes including reading (Reichle et al., 1998), visual attention (Hoffman, 2016), memory (Theeuwes et al., 2009), as well as complex cognitive constructs and state of minds such as situational awareness (Moore & Gugerty, 2010), mental workload (May et al., 1990) or mental fatigue (Bafna & Hansen, 2021). Eye tracking is not only limited to highly controlled studies in the laboratory; it is also used for practical applications involving diverse user profiles, including psychologists, designers, marketing professionals, medical doctors, computer scientists and data scientists who use eye tracker as a data source (e.g. Pupil labs, Tobii, EyeLink, ETVision). Eye tracking is also becoming an object of real-time interaction, for example with video games such as Ubisoft's Assassin's Creed Valhalla and The Division that use head and eye movements for in-game controls (e.g., changing the field of view).

Despite the wide range of eye tracking applications, there are common methodological steps and data analysis pipelines related to gaze analysis (Duchowski, 2017). These include parameters for fixation determination (fixation time, angular velocity), calibration, head movement realignment (for head-mounted eye trackers), and the analysis of certain classical eye metrics (fixation times on areas of interest, visual pattern analysis, etc.). The definition of new types of analysis and new gaze metrics (e.g., visual entropy) is an extremely active area, however, based on our knowledge, no software has an extensible and customizable real time data analysis pipeline neither easy solution to add new gaze metrics proposed in the literature. Indeed, proprietary eye tracking software are generally closed and does not allow for the addition of gaze metrics such as entropy (Krejtz et al., 2014), K-coefficient analysis (Krejtz et al., 2016), explore/exploit ratio (Dehais et al., 2015), N-gram (C. Lounis et al., 2021) or LZC (C. Lounis et al., 2020). These gaze metrics are rarely or not at all available in the manufacturers' eye tracker

software and are still reserved for researchers who manipulate mathematical tools. Even for basic metrics such as the calculation of fixations, multiple algorithms exist in the literature (Salvucci & Goldberg, 2000), and a Tobii Pro Lab user might want to work with another method than those proposed (I-DT or I-VT).

Overall, there are two types of eye-tracking devices, screen-based and head-mounted eye trackers. Each type has its own pros and cons, and the choice depends on the research applications. Screen-based eye trackers (e.g., Tobii Pro Spectrum) are particularly well-suited for studying computer browsing, reading text, or playing video games. Head-mounted eye trackers offer greater mobility, providing users with the freedom to move around their environment naturally, especially with models that use wireless connections (e.g., Pupil Invisible from Pupil Labs, Tobii Pro Glasses 3). They are versatile and can be employed in various contexts, including laboratory studies, field tests, or everyday activities. For instance, recording an individual reading a physical newspaper or a pilot consulting a paper checklist in the cockpit would not be possible with a screen-based eye tracker, as the paper documents would obstruct the eye tracking camera placed in front of the participant. However, head-mounted eye trackers are less accurate than their screen-based counterparts and might be more affected by lighting conditions. Moreover, the gaze position is superimposed on the image taken by a front scene camera seeing the scene, which may become blurred due to head movements. It is also worth noting that the front scene camera resolution (e.g., 1920x1080 for Tobii Pro Glasses 3, 1088x1080 for Pupil Invisible) can make details of the scene being analyzed unreadable. Due to head movement, the analysis of areas of interest requires a supplementary step of head movement realignment to perform the mapping between the view camera and a fixed reference. Such augmented reality feature is achieved by image processing on the view camera image, in which feature points or fiducial markers ((Garrido-Jurado et al., 2016)) are detected. A special case is eye-tracking in multi-screen environments or complex three-dimensional scenes. Head-mounted eye trackers can struggle to understand the depth

of the fixation point. Specific eye trackers based on multiple cameras exist, but their implementation can be complex and these systems are typically expensive (see for example the SmartEye Pro integrated in a flight simulator cockpit, SmartEye, 2024).

In this paper, we describe ARGAZE, an open and flexible software library that provides a unified and modular approach for real-time and/or post-processing gaze analysis applications for either screen-based or head-mounted eye tracking systems. It provides a wide variety of gaze metrics and allows free extension to other ones. Below, we list existing software and their capabilities, then we will detail the ARGAZE software library and its design rationale. Finally, we present three use cases in which ARGAZE is used to process eye data from screen-based or head-mounted eye trackers in real time and in post-processing.

State of the art

Following on from the state of the art carried out by PyTrack (Ghose et al., 2020) and EyeMMV (Krassanakis et al., 2014), we have listed the available gaze analysis software according to two main criteria groups. Our goal is to know if each software is able to handle the various situations we encountered in HMI (Human-Machine Interface) interaction and Human Factor fields.

We have evaluated 11 software products based on available manufacturer/developer documentation (they may not be exhaustive):

- **Tobii**: TobiiProLab, 2024b, TobiiProLab, 2024c,
- **PupilLabs**: PupilLabs, 2024a, PupilLabs, 2024f,
- **Gazealytics**: Chen et al., 2023, Gazealytics, 2024b, Gazealytics, 2024a,
- **PyGaze**: Dalmaijer et al., 2014, PyGaze, 2024,
- **PyTrack**: Ghose et al., 2020, PyTrack, 2024a, PyTrack, 2024b,
- **OGAMA**: Voßkühler et al., 2008, OGAMA, 2024a, OGAMA, 2024b,
- **BeGaze**: Begaze, 2014,
- **EyetrackingR**: Forbes et al., 2023,

- **iMotions:** iMotions, 2024c, iMotions, 2024a, iMotions, 2024b,
- **EyeMMV:** Krassanakis et al., 2014, EyeMMV, 2024,
- **Gazepoint:** Gazepoint, 2024b, Gazepoint, 2024a.

A position paper (Jayawardena, 2022) focuses on the development of a real-time eye-tracking data analysis pipeline. This work is currently in progress, and few details are provided, so we have not included it in the comparison.

Environmental assessment

Table 1 compares the possibilities offered by several gaze analysis software for screen-based and head-mounted devices. Each line in the table is a software available on the market, the color code detailed in the legend shows if it's usable for real-time or/and post-processing. The columns illustrate the variety of uses for each software.

The *Device* columns shows if the software is compatible with screen-based or head-mounted devices.

The *Interface* columns show how the user interacts with the software to build a custom gaze analysis pipeline:

- **Library:** the software is a suite of programming code that is used to develop software programs.
- **API:** the software provides an Application Programming Interface.
- **GUI offline:** the software provides an offline Graphic User Interface.
- **GUI online:** the software provides an online Graphic User Interface.

The *Integration* columns show how it is possible to integrate the software:

- **Data agnostic:** the software works with any gaze data format and works with any eye tracker devices.

- **Free:** the software is free to use, change and distribute with some conditions (e.g., BSD, MIT, etc. licenses)

- **Multi-platform:** the software works on Windows, MacOS and Linux platforms.
- **Extensible:** the software gaze metrics are extensible thanks to a plugin

architecture.

The *Mapping* columns show if the software can process eye tracking data from a screen or the environment:

- **Screen:** the software maps gaze positions from a screen-based eye tracker onto screen objects like images, text, etc.
- **Environment:** the software maps gaze positions from a head-mounted eye tracker device onto environment objects thanks to technologies like 3D scanning, fiducial markers for augmented reality, snapshot mapping, face recognition, etc.

Output assessment

Table 2 lists what kind of gaze analysis each software provides, either for real-time or post-processing. The color code for each line is the same as Table 1.

The *Features and metrics* columns show what kind of analysis the software proposes:

- **Primary:** the software extracts at least fixations and saccades, and possibly blinks, drift, micro-saccades, pupil size, etc.
- **Basic:** the software extracts metrics about numbering, summing, averaging values related to fixations, saccades and Areas of Interest (AOI) (e.g., dwell time, AOI distribution, time to first contact, etc.).
- **Spatial:** the software extracts metrics about the scan path inside a frame reference system (e.g., explore/exploit ratio, nearest neighbor index, K-coefficient, etc.).
- **Sequential:** the software extracts metrics about AOI sequences (e.g., transition matrix, entropy, Lempel-Ziv complexity, N-grams, etc.).
- **Behavioral:** the software extracts metrics about participant behavior (e.g., reading, workload index, fatigue, etc.).

The *experiment* columns show the type of analysis it is possible to do with the software:

- **Statistical:** the software conducts statistical analysis between experimental conditions and different groups of participants.

- **Temporal:** the software extracts metrics according to temporal considerations (e.g., Time Of Interest (TOI), event management, interval-based or binned metrics, etc.).

The *output format* columns show how it is possible to export data from the software:

- **Data:** the software can export analysis data to any file format or publish them thanks to any middleware.

- **Visualization:** the software provides analysis data in a graphical way (e.g., scan path, heatmap, bee swarm, clusters, etc.).

Current shortcomings

Our examination of the various software reveals that none of them fulfills all requirements at once. While some of them are dedicated to screen-based devices only (Gazealytics, Chen et al., 2023, Pytrack, Ghose et al., 2020, OGAMA, Voßkühler et al., 2008, EyeMMV, Krassanakis et al., 2014 and Gazepoint), others aren't addressing real-time (Begaze, 2014, EyetrackingR, Forbes et al., 2023). Besides, the gaze metrics extensibility criterion is almost never envisioned. Namely, *iMotions* (iMotions, 2024c) seems to be the one that offers the most features and allows users to extend analytics with R-Notebooks (iMotions, 2024d) for post-processing, but it is not working on all major OS and is not adapted to all finances. *PupilLabs* is the most flexible software but doesn't provide as many gaze metrics as *iMotions*.

Considering software which are supporting head-mounted devices, it appears that gaze mapping onto a reference image is one of the most common solution (TobiiProLab, 2024a, PupilLabs, 2024e, iMotions, 2024b, Begaze, 2014). *PupilLabs* also provides real-time surface tracking thanks to augmented reality techniques via the use of ArUco marker detection (PupilLabs, 2024d, 2024g) without connecting those data to a real-time advanced gaze analysis pipeline (PupilLabs, 2024c). Further, *PupilLabs* explores post-processing gaze data mapping onto a 3D model of the environment thanks to advances in camera based 3D scene synthesis techniques (PupilLabs, 2024b). Finally, no software integrated an advanced gaze analysis pipeline with real-time gaze mapping onto head-mounted camera images.

Having an open-source, multi-platform and extensible software to face various experimental uses cases, either for real-time or post-processing screen-based or head-mounted gaze data analysis with environment mapping would provide great benefits to many research fields. This is the overall objective of ArGaze.

ArGaze software library

Design foreword

Considering ARGAZE main objective, the design of a reusable software library that covers a wide range of uses needs an abstraction process that cannot be achieved from one single application development (Roberts, Johnson, et al., 1996). ARGAZE has been refined through numerous HMI experiments, including real-time eye tracking interaction, and extensive human factor studies. These experiences have elucidated the various requirements associated with analysis, particularly in terms of eye-tracking data processing and the utilization of advanced gaze metrics. Additionally, the design of the augmented reality gaze mapping pipeline was driven to generalization by the different workspaces where experiments were conducted, such as the multi-screen setups used by air traffic controllers or aircraft cockpits.

According to the identified shortcomings in the currently available software, ARGAZE provides gaze analysis support with the following design rationale:

- to support any screen-based or head-mounted eye trackers: a specific connector adhering to a generic gaze data interface can be developed for each eye tracker model;
- to provide an adaptable gaze analysis pipeline that can be tailored for post- or real-time gaze data processing, recording, and visualization ;
- at each step of the gaze analysis pipeline, to provide ways for third-party developers to add new metrics or algorithms according to a generic classes interface;
- to encourage external contributions with open-sourced code and documented features.

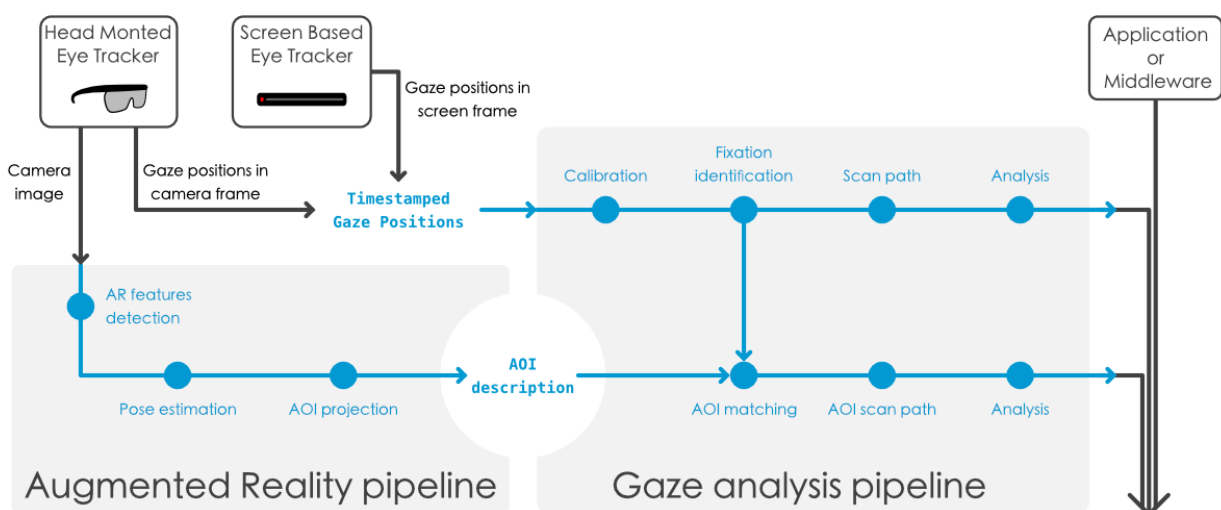
ARGAZE also provides gaze mapping support, extending the previous design rationale as follows:

- to map head-mounted eye tracker gaze data onto a complex and dynamic environment geometry in real-time;
- to project gaze data into screen coordinates system to match with screen objects.

Figure 1 illustrates where to insert the ARGAZE pipeline inside a gaze analysis workflow and how it can be combined with augmented reality gaze mapping in a head-mounted device case.

Figure 1

ArGaze pipeline overview



Computing environment

ARGAZE is implemented with Python (Oliphant, 2007), a computing language popular among scientists and engineers. As a prototyping environment, Python is convenient as it is portable, mixable and free. As a programming language, Python is also powerful as it is Object-Oriented, functional, and relatively easy to use and learn (Lutz, 2013).

Distribution

ARGAZE is released under the GNU General Public License (version 3 Foundation, 2007) and is available on a public read-only git repository at <https://gitpub.recherche.enac.fr/argaze>. Requests or patches can be sent to argaze-contact@recherche.enac.fr. The documentation is available at <http://achil.recherche.enac.fr/features/eye/argaze/index.html>.

Dependencies

ARGAZE is based on Python version 3 and requires the installation of external packages such as NUMPY, SCIPY and PANDAS dedicated to high-performance scientific and data computing (Fuhrer et al., 2021). Besides, ARGAZE requires OPENCV for image processing and object identification to map wearable eye tracker gaze positions onto head-mounted camera (Howse, 2013). OPENCV is also used as default graphical library to draw ARGAZE internal states.

Hardware requirements

ARGAZE has been developed and tested on the following operating systems: Windows 10, Ubuntu 20.04 LTS and MacOS 14. As ARGAZE is written solely in Python and does not use its own compiled code, its portability depends on the availability of all dependencies on a specific system.

The OPENCV package comes with hardware requirements, as it uses the GPU to enable high-rate image processing thanks to CUDA API on NVIDIA devices (Cervera, 2020). ARGAZE has been tuned to work on a NVIDIA Jetson Xavier (30W) with TobiiGlasses front scene camera in realtime (25 FPS 1920x1080).

Gaze analysis pipeline architecture

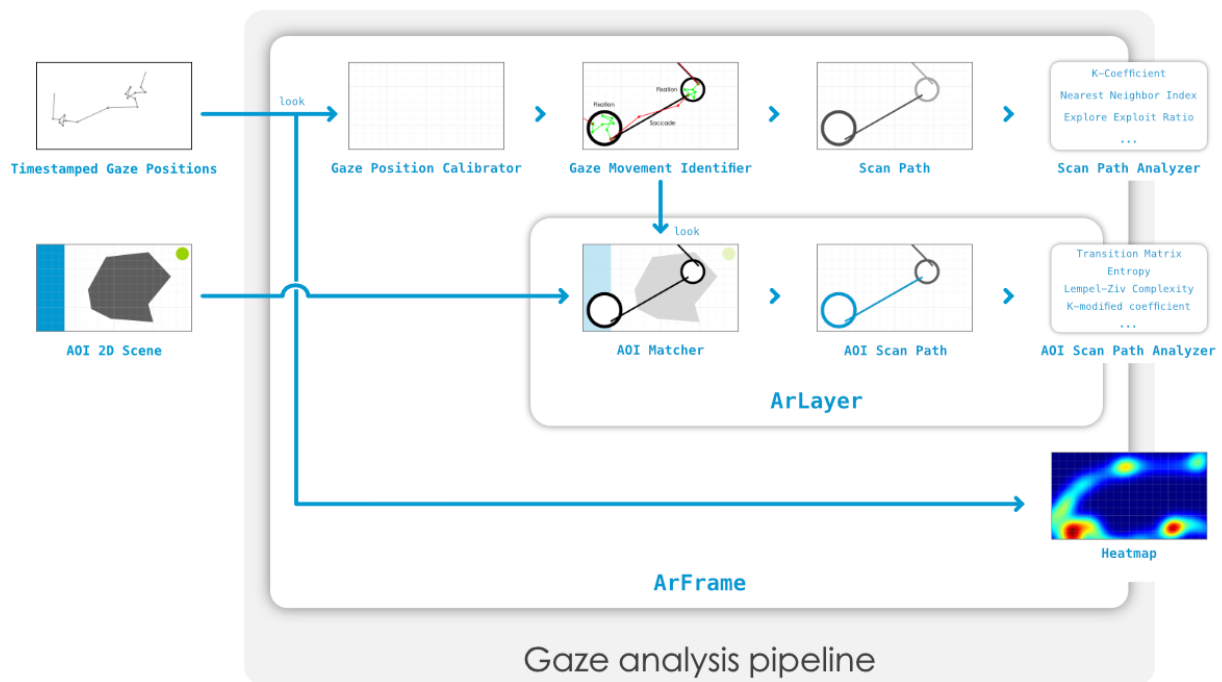
A gaze analysis pipeline can be designed as a white box library (Johnson & Foote, 1988) where each pipeline step is generalized by an abstract class. White box design pattern allows any library user to create new classes by inheritance and overriding methods

that are different in each subclass.

ARGAZE provides an extensible modules library allowing to select application-specific algorithms at each pipeline step (Figure 2) like for fixations and saccades identification, AOI matching or scan path analysis. Once incoming data is formatted as required, all those gaze analysis features can be used with any screen-based eye tracker device.

Figure 2

Gaze analysis pipeline overview



Timestamped gaze positions as context

Whatever eye data comes from a file on disk or from a live stream, timestamped gaze positions are required to feed the gaze analysis pipeline. For the post-processing use case, timestamped gaze positions can be loaded from a CSV file thanks to a PANDAS `DataFrame` object before being stored into a `TimestampedGazePositions` class. When gaze positions come from a real-time input, each single gaze position can be edited thanks to the

`GazePosition` class. Timestamps can either be integers or floats, seconds, milliseconds or whatever is needed. The only concern is that all time values used in further pipeline steps have to be in the same unit.

Specific eye-tracker device connectors can be designed thanks to `ArContext` class. This interface makes sure that eye-tracking data is sent to a gaze analysis pipeline using Python context management, which helps finish the process properly. `ArContext` class also provides pipeline execution time assessment and exception management.

Rectangular frame as main interface

ARGAZE defines `ArFrame` class as a rectangular area where timestamped gaze positions are projected in and inside which they need to be analyzed. Gaze positions have to be in the same range as rectangular area dimensions to be projected in and can either be integers or floats, pixels, millimeters or whatever is needed. The only concern is that all spatial values used in further pipeline steps have to be in the same unit.

Considering the gaze position calibration step is optional, the first mandatory `ArFrame` pipeline step is to identify fixations or saccades¹ from consecutive timestamped gaze positions. The identification algorithm can be selected by instantiating a particular `GazeMovementIdentifier` from the `GAZEANALYSIS` submodule listed in Appendix A or from another Python package.

The second `ArFrame` pipeline step aims to build a `ScanPath` defined as a list of `ScanSteps` made by a fixation and a consecutive saccade. Once fixations and saccades are identified, they are automatically appended to the `ScanPath`. A `ScanPath` have a `duration_max` attribute to set the duration from which older scan steps are removed each time new scan steps are added, allowing to set up sliding window analysis.

The last `ArFrame` pipeline step consists of passing the built `ScanPath` to each loaded `ScanPathAnalyzer`. Each analysis algorithm can be selected by instantiating a particular `ScanPathAnalyzer` from the `GAZEANALYSIS` submodule listed in Appendix A or from

¹ In ARGAZE, `Fixation` and `Saccade` are considered as particular `GazeMovement`.

another Python package.

AOI management as frame layers

Besides, ARGAZE also defines `ArLayer` class as a space where to make matching of fixations with a set of 2D AOI and inside which those matchings need to be analyzed.

The first `ArLayer` pipeline step aims to match identified gaze movement with layers AOI. The matching algorithm can be selected by instantiating a particular `AOIMatcher` from the `GAZEANALYSIS` submodule listed in Appendix A or from another Python package.

The second `ArLayer` pipeline step aims to build an `AOIScanPath` defined as a list of `AOIScanSteps` made by a set of successive fixations/saccades onto the same AOI. Once gaze movements are matched to AOI, they are automatically appended to the `AOIScanPath`. An `AOIScanPath` has a `duration_max` attribute to set the duration from which older AOI scan steps are removed each time new AOI scan steps are added, allowing to set up sliding window analysis.

The last `ArLayer` pipeline step consists in passing the built `AOIScanPath` to each loaded `AOIScanPathAnalyzer`. Each analysis algorithm can be selected by instantiating a particular `AOIScanPathAnalyzer` from the `GAZEANALYSIS` submodule listed in Appendix A or from another Python package.

An `ArFrame` instance can contains multiples `ArLayers` instances in order to separate AOI analysis according to any experiment criteria.

Configuration and execution

`ArFrame` instance creation is done by loading a JSON configuration file to customize each pipeline step. Then, to execute the whole instantiated pipeline, timestamped gaze positions have to be passed one by one to the `ArFrame` instance (Code Listing 1).

```
1 import argaze
2
3 # Load ArFrame from a JSON configuration file
```



```
4 with argaze.load('./configuration.json') as ar_frame:
5
6     # Assuming that timestamped gaze positions are available
7     ...
8     # Execute ArFrame for a timestamped gaze position
9     ar_frame.look(timestamped_gaze_position)
```

Code Listing 1: Gaze analysis pipeline loading and execution

Recording

ArFrame and ArLayer analysis can be recorded by registering observers to their *look* method. Both classes get an `observers` attribute, allowing them to load classes from any Python package. The Code Listing 2 illustrates how to create an ArFrame *look* method observer that will write desired ArFrame data into a CSV file.

```
1 from argaze.utils import UtilsFeatures
2
3 # Define a custom recorder class to observe look method call
4 class MyFrameRecorder(UtilsFeatures.FileWriter):
5
6     def __init__(self, **kwargs):
7
8         # Init FileWriter class
9         super().__init__(**kwargs)
10
11        # Edit header line
12        self.header = "Timestamp (ms)", ...
13
14    def on_look(self, timestamp, ar_frame, exception):
15
16        # Record any ArFrame data
17        data = (
18            timestamp,
```

```
19         ...
20     )
21
22     # Write to file
23     self.write(data)
```

Code Listing 2: Gaze analysis recording

`ArFrame` and `ArLayers` observers are automatically notified after each `ArFrame` pipeline execution.

Visualization

Many `ArFrame` and `ArLayers` internal states can be drawn in real time or afterward, depending on the application purpose. Image parameters can be loaded from a JSON configuration file or set up at runtime.

Optional pipeline steps

Gaze position calibration, heat map and background image are optional `ArFrame` pipeline steps to set up at configuration time. There are usually processed each time the `ArFrame` pipeline is executed and, if it is relevant, the resulting image is accessible provided that it is requested in the image parameters.

Scripting pipeline access

All gaze analysis pipeline internal states are accessible from a Python script. This could be particularly useful for real-time gaze interactions.

First of all, the `ArFrame` configuration can be loaded from a Python dictionary. Then, once the configuration is loaded, it is possible to access its attributes as described in the code reference documentation. Thus, each `ArFrame` layer is accessible by name and so, access to their attributes too.

In addition, `ArFrame` execution returns many data, like calibrated gaze position, identified gaze movement, scan path analysis, layers AOI scan path analysis, execution times and exceptions that can be exploited for real-time interactions.

External gaze analysis libraries

As detailed in Appendix A, the `GazeAnalysis` submodule contains many algorithms available to start using the ARGAZE software library. However, it is possible to load `GazeMovementIdentifier`, `ScanPathAnalyzer`, `AOIMatcher`, `AOIScanPathAnalyzer` or `GazePositionCalibrator` modules from other Python packages.

The writing of new modules consist to inherit from the abstract classes above, writing each expected member methods according to the given interface, and then to import the package. For instance, in order to implement a new gaze movement identification algorithm since ARGAZE only provides I-DT and I-VT, a contributor should consider writing a new Python file as illustrated in CodeListing 3. Once specific Fixation and Saccade classes have been defined, the contributor has to write a specific `GazeMovementIdentifier` class with an *identify* method and some required properties. The *identify* method requires a timestamped gaze position from where identification has to be done considering former timestamped gaze positions and an optional *terminate* parameter to notify the identification algorithm that the given gaze position will be the last one. The *identify* method should return identified gaze movement once it is finished otherwise, it returns empty gaze movement at least. It is mandatory that each identified gaze movement share its first/last gaze position with the previous/next gaze movement.

```

1 from argaze import GazeFeatures, DataFeatures
2
3 class Fixation(GazeFeatures.Fixation):
4     """Define algorithm specific fixation."""
5     ...
6
7 class Saccade(GazeFeatures.Saccade):
8     """Define algorithm specific saccade."""
9     ...
10
11 class GazeMovementIdentifier(GazeFeatures.GazeMovementIdentifier):

```

```
12     """Implementation of a specific identification algorithm."""
13
14     @DataFeatures.PipelineStepInit
15     def __init__(self, **kwargs):
16         """Initialize identification algorithm."""
17
18         # Init GazeMovementIdentifier class
19         super().__init__()
20         ...
21
22     @DataFeatures.PipelineStepMethod
23     def identify(self, timestamped_gaze_position: GazePosition, terminate=
False):
24         """Identify gaze movement from successive timestamped gaze
positions."""
25         ...
26
27     def current_gaze_movement(self):
28         """Get the current identified gaze movement (finished or in
progress) if it exists otherwise, an empty gaze movement."""
29         ...
30
31     def current_fixation(self):
32         """Get the current identified fixation (finished or in progress)
if it exists otherwise, an empty gaze movement."""
33         ...
34
35     def current_saccade(self):
36         """Get the current identified saccade (finished or in progress) if
it exists otherwise, an empty gaze movement."""
37         ...
```

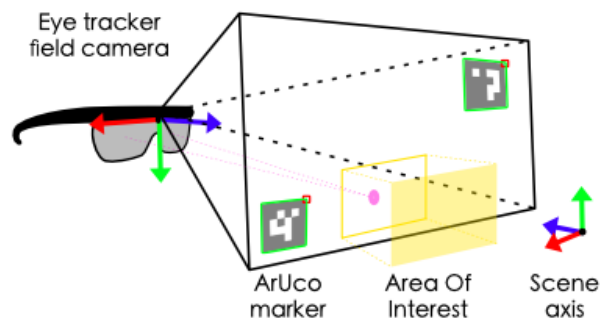
Code Listing 3: Gaze analysis new module

Addressing real time gaze mapping in complex geometry workspace

Detection of square-based fiducial markers for camera pose estimation is a fast and robust solution that is very popular in robot navigation or augmented reality application (Garrido-Jurado et al., 2016). As proposed by Duchowski et al., 2020, ARGAZE provides augmented reality support to map AOI on ARUCO markers set up (Figure 3) and extend gaze analysis pipeline to complex geometry workspace.

Figure 3

ArUco pipeline axis



Camera frame as main interface

ARGAZE defines an abstract `ArCamera` class to handle gaze mapping on the basis of any augmented reality technology. As an `ArFrame` child, the `ArCamera` class benefits from all the services described in section Gaze analysis pipeline architecture. Besides, ARGAZE defines an abstract `ArScene` class to describe and project `ArLayers` and `ArFrames` into an `ArCamera` on the basis of any augmented reality technology. An `ArCamera` instance can contain multiple `ArScenes` instances in order to handle several independent scenes inside the same workspace according to any experiment criteria.

`ArCamera` class defines an abstract `watch` method to detect augmented reality features from camera images, estimate the pose of scenes and, then project them into the camera frame. Then, as `ArCamera` inherits from `ArFrame`, its gaze analysis pipeline can be executed to project gaze positions into projected layers inside the camera frame. An internal `ArCamera` thread locker delays `ArCamera` gaze analysis pipeline execution while

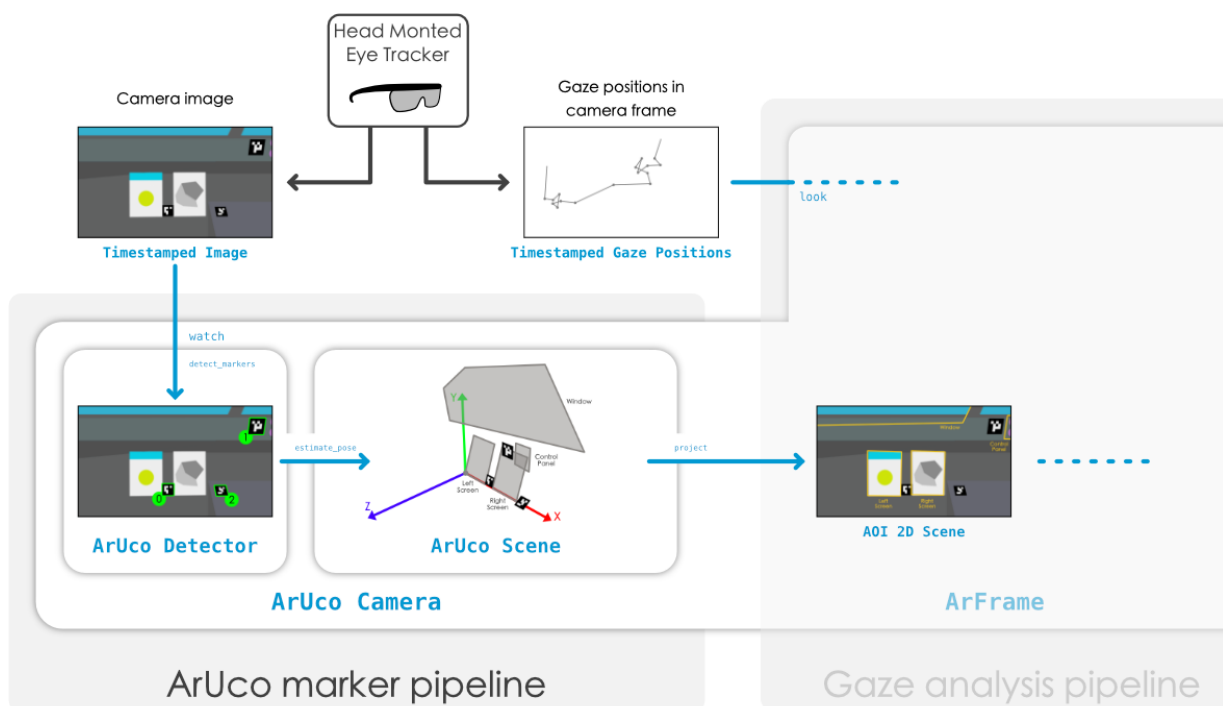
ArCamera is processing a camera image, allowing parallel processes to share the same ArCamera instance.

ArUco marker technology as a submodule

As a specific augmented reality technology, ARUCO markers are embedded in ARGAZE as dedicated submodule where ArUcoCamera inherits from ArCamera class and ArUcoScene inherits from ArScene class.

Figure 4

ArUco marker pipeline overview



As illustrated in Figure 4, the first ArUcoCamera pipeline step is to detect all markers from a specific ARUCO dictionary² with a given size inside input image thanks to an ArUcoDetector instance.

The second ArUcoCamera pipeline step is to estimate scenes pose then to project each scenes layers into corresponding camera frame layers. To make it possible, each

² An ARUCO marker belongs to a set of markers called ARUCO dictionary.

`ArUcoScene` instance have to describe the places of each ARUCO marker and each layer AOI into the same 3D world referential.

An optional `ArUcoCamera` pipeline step (not illustrated in Figure 4) allows to project each scenes frames layer into corresponding scene layers then, into corresponding camera frame layers. To make it possible, each `ArUcoScene` instance has to describe each frame layer AOI into a 2D frame referential.

This architecture allows projecting several scenes, each related to an ARUCO markers subset, that can be spatially independent, like a static desktop screen and a movable tablet screen with their own AOI inside each screen referential.

Furthermore, `ArUcoCamera` creation is also done by loading a JSON configuration file to customize each pipeline step. `ArUcoCamera` image defines extra parameters to visualize ARUCO marker detection, scene projections or optic parameters. The Code Listing 4 illustrates how to load and execute `ArUcoCamera` pipeline.

```
1 import argaze
2 from argaze.DataFeatures import TimestampedImage
3
4 # Load ArUcoCamera from a JSON configuration file
5 with argaze.load('./configuration.json') as aruco_camera:
6
7     # Assuming that a video stream or file is opened
8     ...
9
10    # Assuming that the video reading is handled in a dedicated thread
11    ...
12
13    # Capture timestamp and image from video stream of file
14    timestamp, image = video_capture.read()
15
16    # Edit timestamped image
17    timestamped_image = TimestampedImage(image, timestamp=timestamp)
```

```
18     # Detect ArUco markers, estimate scenes pose then, project them
    into camera frame
19     aruco_camera.watch(timestamped_image)
20
21     # Assuming that timestamped gaze positions are available in another
    thread
22     ...
23     # Execute ArUcoCamera frame for a timestamped gaze position
    aruco_camera.look(timestamped_gaze_position)
24
25
26     # Assuming that visualization is running in main thread
    ...
27
28     # Display ArUcoCamera frame image.
    ... aruco_camera.image()
```

Code Listing 4: ARUCO markers pipeline execution

Cases studies

Here we present a number of studies that have used the ARGAZE software library. These studies used either head-mounted or screen-based eye trackers, and the data was processed online or post-processed. This gives an overview of the different possibilities offered by the ARGAZE software library.

Post-processing head-mounted eye tracking data

Following use case has integrated ARGAZE ArUco markers to map air traffic controllers gaze onto multiple screens environment in post-processing then, enable scan path study thanks to gaze analysis pipeline.

Experiment background

The next-gen air traffic control system (4Flight) aims to enhance the operational capacity of the en-route control center by offering new tools to air traffic controllers. However, it entails significant changes in their working method, which will consequently

have an impact on how they are trained. Several research projects (Kang and Landry, 2014, Palma Fraga et al., 2021, Wang et al., 2021) on visual patterns of air traffic controllers indicate the urgent need to improve the effectiveness of training in visual information seeking behavior. An exploratory study was initiated by a group of trainee air traffic controllers with the aim of analyzing the visual patterns of novice controllers and instructors, intending to propose guidelines regarding the visual pattern for training.

Experiment environment

The 4Flight control position (Figure 5) consists of two screens: the first displays the radar image along with other information regarding the observed sector, the second displays the agenda, which allows the controller to link conflicting aircraft by creating data blocks, and the Dyp info, which displays some information about the flight. During their training, controllers are taught to visually follow all aircraft streams along a given route, focusing on their planned flight path and potential interactions with other aircraft.

Figure 5

4Flight working position with ArUco markers

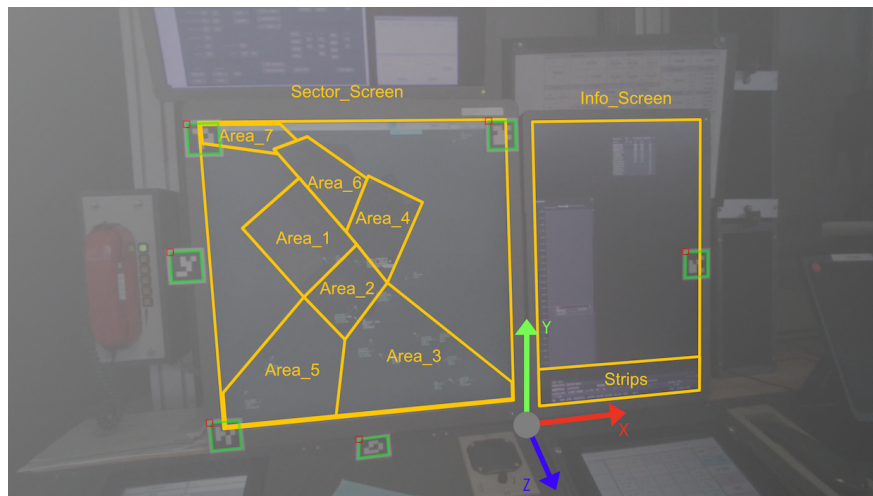


A traffic simulation of moderate difficulty with a maximum of 13 and 16 aircraft simultaneously was performed by air traffic controllers. The controller could encounter lateral conflicts (same altitude) between 2 and 3 aircraft and conflicts between aircraft that need to ascend or descend within the sector. After the simulation, a directed interview

about the gaze pattern was conducted. Eye tracking data was recorded with a Tobii Pro Glasses 2, a head-mounted eye tracker. The gaze and scene camera video were captured with Tobii Pro Lab software and post-processed thanks to ARGAZE software library in a Python script. As the eye tracker model is head mounted, ArUco markers were placed around the two screens to ensure that several of them were always visible in the field of view of the eye tracker camera (Figure 6).

Figure 6

AOI and ArUco marker 3D description



Various metrics were calculated considering the entire size of the two screens, like average fixation duration, explore/exploit ratio, and K-coefficient, AOI distribution, transition matrix, entropy and N-grams considering this time the different AOIs of the environment (sectors and UI panels). All analyses were exported thanks to ARGAZE recording system.

Real time head-mounted eye tracking interactions

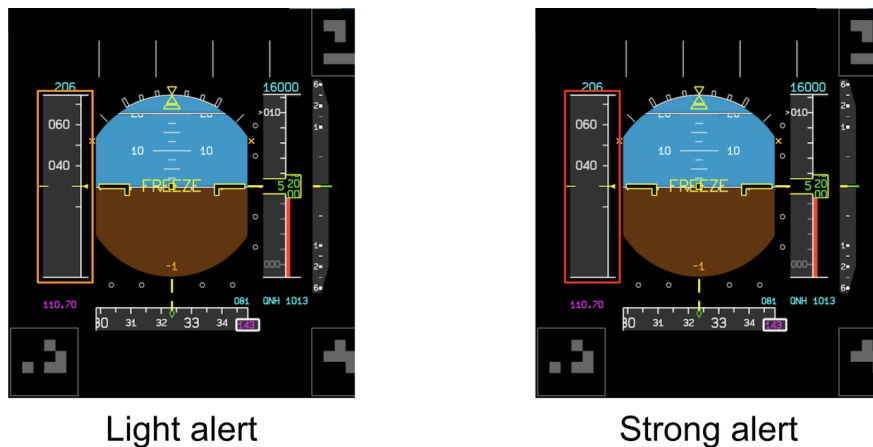
The following use case has integrated ARGAZE ArUco marker pipeline to map pilot gaze onto many cockpit instruments in real-time and then enable fixation matching thanks to the gaze analysis pipeline. ArGaze enabled a cognitive assistant to support a pilot's situation awareness.

Experiment background

The HAIKU project (Haiku, 2023) aims to pave the way for human-centric intelligent assistants in the aviation domain by supporting, among others, the pilot during startle or surprise events. One of the features provided by the assistant through ARGAZE is a situation awareness support that ensures the pilot updates his awareness of the aircraft state by monitoring his gaze and flight parameters. When this support is active, relevant information is highlighted on the Primary Flight Display (PFD; see Figure 7).

Figure 7

Haiku assistant PFD highlights



Experiment environment

Pilot eye tracking data were provided by Tobii Pro Glasses 2, a head-mounted eye tracker. The gaze and scene camera video were captured through the Tobii SDK and processed in real-time on NVIDIA Jetson Xavier thanks to ARGAZE software library in a Python script (Figure 8). Since the eye tracker model is head-mounted, ArUco marker were placed at various locations within the A320 cockpit to ensure that several of them were constantly visible in the field of view of the eye tracker camera (Figure 8).

Figure 8

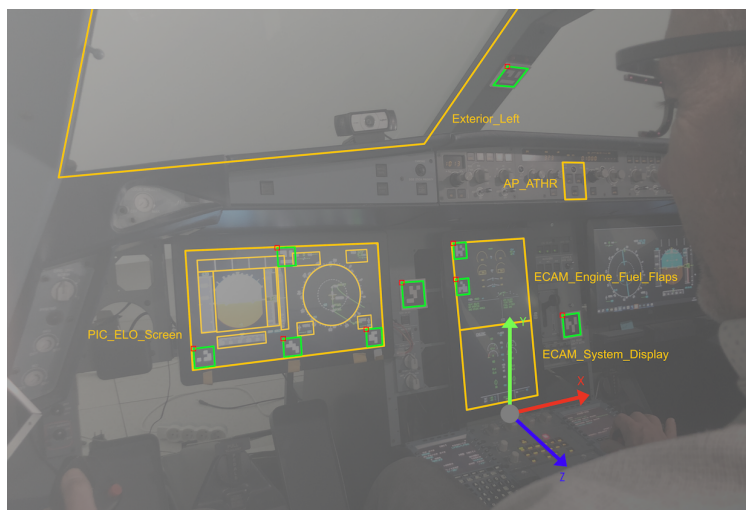
Airbus A320 simulator cockpit with ArUco markers



The ArUco marker pipeline has enabled real-time gaze mapping onto multiple screens and panels around pilot-in-command position while gaze analysis pipeline was identifying fixations and matching them with dynamic AOIs related to each instruments. To identify the relevant AOIs, a 3D model of the cockpit describing the AOI and the position of the markers has been realized (Figure 9).

Figure 9

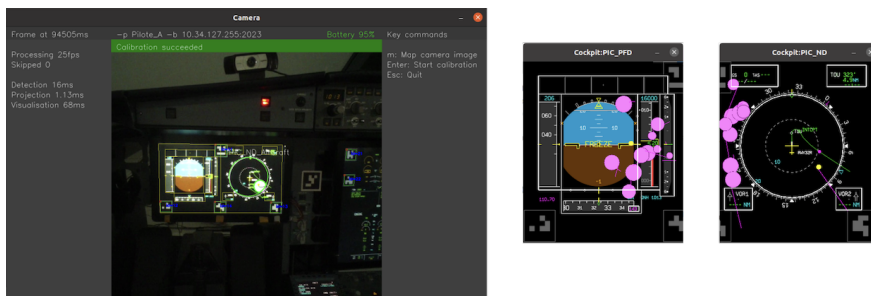
AOI and ArUco markers 3D description



ARGAZE recording system was delivering the fixations on the Ivy bus middleware (Buisson et al., 2002) for theAOI, which were then used by the situation awareness agent. A GUI was showing the mapping for monitoring purposes (Figure 10).

Figure 10

Haiku FOCUS assistant gaze mapping visualization



Experiment results

Although it is not possible to report statistical analysis due to the sample size of the study (5 pilots), participants were generally positive about using the assistant and found it particularly useful in manual piloting, as it allowed them to maintain good situational awareness and the awareness guidance was generally relevant.

Post-processing screen-based eye tracking data

The following use case has integrated ARGAZE gaze analysis pipeline to post-process eye tracking data to export basic and spatial metrics.

Experiment background

The SELICNA project aimed to understand the relationship between inter-individual differences of trainees and success in air traffic control training. In a selection context, these inter-individual differences are usually cognitive abilities assessed by psychometric tests (Mouratille et al., 2022). However, the measurement of perceptual efficiency may be relevant. It can be tested by modulating task difficulty and measured with an eye tracker.

Experiment environment

Twenty-six air traffic controllers were recruited at the beginning of their training period. Participant eye tracking data were recorded with a screen-based Tobii Spectrum eye tracker (600Hz, Figure 11). The gaze and pupil were collected through the Tobii SDK and post-processed with the ARGAZE software library in a Python script. No ArUco marker was used since coordinates of the screen are known by the screen-based eye tracker. After a baseline period of five minutes, participants had to perform a dual-task paradigm. Two concurrent tasks, an ATC task and a N-back task, were integrated into an adapted version of a low-fidelity ATC simulation named Labyrinth microworld (Imbert et al., 2014). Difficulty level was manipulated within the N-back task.

Figure 11

SELICNA project : Experimental environment

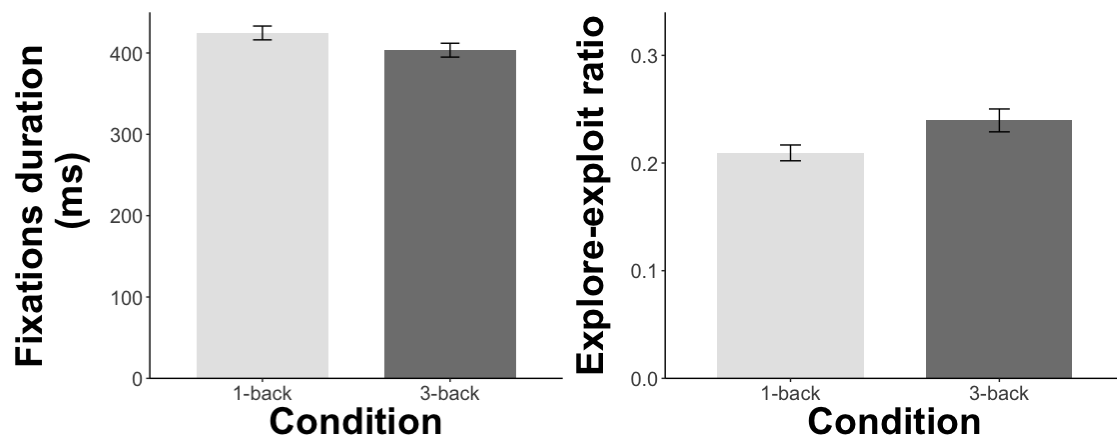


Experiment results

ARGAZE gaze analysis pipeline has enabled the comparison of the pupil diameters to the baseline period and to extract fixations duration (Figure 12a). Spatial metrics have also been calculated such as the Explore-Exploit Ratio (Figure 12b) and K-coefficient.

Figure 12

SELICNA project: some metrics according to conditions

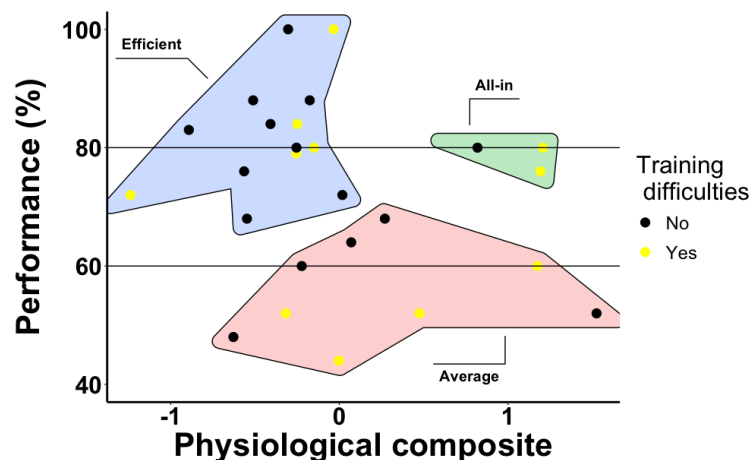
(a) *Fixations duration*(b) *Explore-Exploit Ratio*

A physiological composite score was computed with all these metrics. This composite score and the performance score were classified by a cluster analysis. Three classes were highlighted : an "Efficient" with an excellent performance and a low physiological cost, an "All-in" with an excellent performance and a high physiological cost, and an "Average" with an average performance and a high physiological cost. These classes were used in an exact logistic regression to predict training difficulties (Figure 13).

However, training difficulties could not be predicted by these classes.

Figure 13

SELICNA project: Performance, physiological composite and training difficulties



Perspectives for improvement

Even if ARGAZE addresses many experimental situations involving eye movement analysis, it still has some limitations and many improvements can be envisioned. In Duchowski's "Eye Tracking Methodology" book (Duchowski, 2017), a gaze analysis pipeline consists of several steps that aren't yet all included in ARGAZE gaze analysis pipeline. For instance, ARGAZE does not enable visual angle conversion of raw eye tracking data. The next ARGAZE version will enable the insertion of such step considering display screen dimensions, resolution and viewing distance whether for screen-based or head-mounted devices. It is not possible either to denoise and filter raw eye tracking data. As this would be a major improvement, the next ARGAZE version will enable the insertion of algorithms like Butterworth smoothing, Savitsky-golay differentiation and velocity thresholding.

ARGAZE visualization support could also be improved to help in fine-tuning parameters at each step of the pipeline and to provide detailed experiments results. OpenCV is used as default graphic library to draw ARGAZE objects state. Future developments plan to enable users to select the most suitable graphic engine according to the required drawing performance (like OpenGL) or to use their own. There is no plotting support but this could be enabled in further versions to provide ready-made Matplotlib exportation at each pipeline step.

Comparing gaze analysis algorithms or proving their reliability is very important to consolidate any experiment results. As ARGAZE proposes a unified pipeline, it would be particularly adapted to build bench marking support based on Gold Standard datasets.

Whether through standard image processing or recent machine learning technologies and specifically fast single stage object detectors (.i.e Yolo, Diwan et al., 2023, Shafiee et al., 2017) many objects can be detected in a live video stream, enabling dynamic AOI to be defined for gaze mapping or reference image mapping. It is possible to reconstruct a 3D scene through the detection of invariants in an image, but this comes at a cost. If the scene is specific, it will require specific supervised learning, and tagging images with the AOI or

objects one would like to detect takes time. ARGAZE has been designed in such a way that it is possible to incorporate other types of detection into the pipeline.

Setting up an ARGAZE configuration file and writing the Python script that will load it requires some programming skills. Having a GUI editor to create its own gaze analysis pipeline and even more, describe where AOI are in a 2D or 3D scenes, would be a great tool accessible to non-programmers. Another feature of such GUI would be to create custom dashboard to monitor any processed gaze metrics. ARGAZE is designed to be integrated into such application, even if its development is not planned yet.

Finally, the broaden discussion, maybe the implementation of open eye tracking analysis would facilitate the development of a structured community and advance the adoption of an open science approach for eye tracking.

Acknowledgement

This work was partially done as part of the HAIKU project. This project has received funding from the European Union's Horizon Europe research and innovation programme HORIZON-CL5-2021-D6-01-13 under Grant Agreement no 101075332. The authors wish to acknowledge our beta testers from l'Ecole de l'Air et de l'Espace (Prof. Camachon & Dr. Vantrepotte) and Reims ACC (Mrs Rassel & Mrs Bourgouin).

Table 1*Environmental assessment.*

Topics		Device		Interface				Integration				Mapping	
<i>Softwares</i>	Last update	Screen-based	Head-mounted	Library	API	GUI offline	GUI online	Data agnostic	Free	Multi-platform	Extensible	Screen	Environment
<i>Tobii</i>	2023	■	■	■	■	■	■	■	■	■	■	■	■
<i>PupilLabs</i>	2024	-	■	■	■	■	■	■	■	■	■	-	1
<i>Gazeanalytics</i>	2023	■	■	■	■	■	■	■	■	■	■	■	■
<i>PyGaze</i>	2019	■	■	■	■	■	■	■	■	■	■	■	■
<i>PyTrack</i>	2023	■	■	■	■	■	■	■	■	■	■	■	■
<i>OGAMA</i>	2022	■	■	■	■	■	■	■	■	■	■	■	■
<i>BeGaze</i>	2014	■	■	■	■	■	■	■	■	■	■	■	■
<i>EyetrackingR</i>	2015	■	■	■	■	■	■	■	■	■	■	■	■
<i>iMotions</i>	2024	■	■	2	■	■	3	■	■	■	■	■	■
<i>EyeMMV</i>	2019	■	■	■	■	■	■	■	■	■	■	■	■
<i>Gazepoint</i>	2014	■	■	■	■	■	■	■	■	■	■	■	■

Legend:

■ available for post and real-time processing

■ available for post processing only

■ not available


- not relevant

¹ Available with real-time API combined with *Real-time Screen Gaze* package (PupilLabs, 2024d).² iMotions R library enables notebooks to generate metrics (iMotions, 2024d).³ An online version of iMotions is available for website studies (iMotions, n.d.).


Table 2*Output assessment.*

Topics	Features and metrics					Experiment		Outputs	
	Primary	Basic	Spatial	Sequential	Behavioral	Temporal	Statistical	Data	Visualisation
<i>Softwares</i>									
<i>Tobii</i>									
<i>PupilLabs</i>		1							
<i>Gazeanalytics</i>									
<i>PyGaze</i>									
<i>PyTrack</i>					2				
<i>OGAMA</i>						3			
<i>BeGaze</i>									
<i>EyetrackingR</i>									
<i>iMotions</i>									
<i>EyeMMV</i>									
<i>Gazepoint</i>									

Legend:

 available for post and real-time processing

 available for post processing only

 not available

- not relevant

¹ PupilLabs have tutorials about how to exploit real-time raw gaze data and primary features so it would be possible to build any real-time metrics.

² PyTrack extracts reading behavior.

³ OGAMA has an event database but there is no temporal analysis feature.

References

- Bafna, T., & Hansen, J. P. (2021). Mental fatigue measurement using eye metrics: A systematic literature review. *Psychophysiology*, *58*(6), e13828.
- Begaze. (2014). *Manual*. Version 3.4. SensoMotoric Instruments.
https://psychologie.unibas.ch/fileadmin/user_upload/psychologie/Forschung/N-Lab/SMI_BeGaze_Manual.pdf
- Buisson, M., Bustico, A., Chatty, S., Colin, F.-R., Jestin, Y., Maury, S., Mertz, C., & Truillet, P. (2002). Ivy: A bus software to the development of prototype systems interactive. *ACM International Conference Proceeding Series*, *32*, 223–226.
- Cervera, E. (2020). Gpu-accelerated vision for robots: Improving system throughput using opencv and cuda. *IEEE Robotics & Automation Magazine*, *27*(2), 151–158.
- Chen, K.-T., Prouzeau, A., Langmead, J., Whitelock-Jones, R. T., Lawrence, L., Dwyer, T., Hurter, C., Weiskopf, D., & Goodwin, S. (2023). Gazeanalytics: A unified and flexible visual toolkit for exploratory and comparative gaze analysis. *Proceedings of the 2023 Symposium on Eye Tracking Research and Applications*, 1–7.
- Dalmaijer, E. S., Mathôt, S., & Van der Stigchel, S. (2014). Pygaze: An open-source, cross-platform toolbox for minimal-effort programming of eyetracking experiments. *Behavior research methods*, *46*, 913–921.
- Dehais, F., Peysakhovich, V., Scannella, S., Fongue, J., & Gateau, T. (2015). "automation surprise" in aviation: Real-time solutions. *Proceedings of the 33rd annual ACM conference on Human Factors in Computing Systems*, 2525–2534.
- Di Nocera, F., Terenzi, M., Camilli, M., et al. (2006). Another look at scanpath: Distance to nearest neighbour as a measure of mental workload. *Developments in human factors in transportation, design, and evaluation*, (-), 295–303.
- Diwan, T., Anirudh, G., & Tembhurne, J. V. (2023). Object detection using yolo: Challenges, architectural successors, datasets and applications. *multimedia Tools and Applications*, *82*(6), 9243–9275.

- Drewes, H., Pfeuffer, K., & Alt, F. (2019). Time-and space-efficient eye tracker calibration. *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*, 1–8.
- Duchowski, A. T. (2017). The gaze analytics pipeline. *Eye Tracking Methodology: Theory and Practice*, 175–191.
- Duchowski, A. T., Peysakhovich, V., & Krejtz, K. (2020). Using pose estimation to map gaze to detected fiducial markers. *Procedia Computer Science*, 176, 3771–3779.
- EyeMMV. (2024). *Sources*. Retrieved March 13, 2024, from <https://github.com/krasvas/EyeMMV>
- Forbes, S., Dink, J., & Ferguson, B. (2023). *Eyetrackingr* [R package version 0.2.1]. <http://www.eyetracking-r.com/>
- Foundation, F. S. (2007, June 29). Gnu general public license. *Free Software Foundation*. <http://www.gnu.org/licenses/gpl.html>
- Fuhrer, C., Solem, J. E., & Verdier, O. (2021). *Scientific computing with python: High-performance scientific computing with numpy, scipy, and pandas*. Packt Publishing Ltd.
- Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., & Medina-Carnicer, R. (2016). Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, 51, 481–491. <https://doi.org/https://doi.org/10.1016/j.patcog.2015.09.023>
- Gazealytics. (2024a). *Sources*. Retrieved March 13, 2024, from <https://github.com/gazealytics/gazealytics-master>
- Gazealytics. (2024b). *Web-based application*. Retrieved March 13, 2024, from <https://www2.visus.uni-stuttgart.de/gazealytics/>
- Gazepoint. (2024a). *User manual*. Retrieved March 13, 2024, from <http://andrewd.ces.clemson.edu/courses/cpsc412/manuals/Gazepoint%20Analysis.pdf>

- Gazepoint. (2024b). *Website*. Retrieved March 13, 2024, from <https://www.gazepoint.com/product/gazepoint-analysis-professional-edition-software/>
- Ghose, U., Srinivasan, A. A., Boyce, W. P., Xu, H., & Chng, E. S. (2020). Pytrack: An end-to-end analysis toolkit for eye tracking. *Behavior research methods*, 52, 2588–2603.
- Haiku. (2023). *Haiku project*. Haiku consortium. Retrieved November 12, 2023, from <https://haikuproject.eu/>
- Hoffman, J. E. (2016). Visual attention and eye movements. *Attention*, 119–153.
- Howse, J. (2013). *Opencv computer vision with python* (Vol. 27). Packt Publishing Birmingham.
- Imbert, J.-P., Hodgetts, H. M., Parise, R., Vachon, F., & Tremblay, S. (2014). The laby microworld: A platform for research, training and system engineering. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 58, 1038–1042.
- iMotions. (2024a). *Api*. Retrieved March 13, 2024, from <https://imotions.com/products/imotions-lab/developers/api/>
- iMotions. (2024b). *Gaze mapping*. Retrieved March 13, 2024, from <https://imotions.com/blog/learning/best-practice/gaze-mapping/#gaze-mapping-solves-the-problem>
- iMotions. (2024c). *Lab*. Retrieved March 13, 2024, from <https://imotions.com/products/imotions-lab/>
- iMotions. (2024d). *R-notebooks*. Retrieved March 13, 2024, from <https://imotions.com/products/imotions-lab/developers/r-notebooks/>
- iMotions. (n.d.). *Online*. <https://imotions.com/products/imotions-online/>
- Jayawardena, G. (2022). Introducing a real-time advanced eye movements analysis pipeline. *2022 Symposium on Eye Tracking Research and Applications*, 1–2.
- Johnson, R. E., & Foote, B. (1988). Designing reusable classes. *Journal of object-oriented programming*, 1(2), 22–35.

- Kang, Z., & Landry, S. J. (2014). Using scanpaths as a learning method for a conflict detection task of multiple target tracking. *Human factors*, *56*(6), 1150–1162.
- Krassanakis, V., Filippakopoulou, V., & Nakos, B. (2014). Eyemmv toolbox: An eye movement post-analysis tool based on a two-step spatial dispersion threshold for fixation identification. *Journal of Eye Movement Research*, *7*(1).
- Krejtz, K., Duchowski, A., Krejtz, I., Szarkowska, A., & Kopacz, A. (2016). Discerning ambient/focal attention with coefficient k. *ACM Transactions on Applied Perception (TAP)*, *13*(3), 1–20.
- Krejtz, K., Szmidt, T., Duchowski, A. T., & Krejtz, I. (2014). Entropy-based statistical analysis of eye movement transitions. *Proceedings of the Symposium on Eye Tracking Research and Applications*, 159–166.
- Lounis, C., Peysakhovich, V., & Causse, M. (2020). Lempel-ziv complexity of dwell sequences: Visual scanning pattern differences between novice and expert aircraft pilots. *Eye-Tracking in Aviation. Proceedings of the 1st International Workshop (ETAVI 2020)*, 61–68.
- Lounis, C., Peysakhovich, V., & Causse, M. (2021). Visual scanning strategies in the cockpit are modulated by pilots' expertise: A flight simulator study. *PLoS one*, *16*(2), e0247061.
- Lounis, C. A., Hassoumi, A., Lefrancois, O., Peysakhovich, V., & Causse, M. (2020). Detecting ambient/focal visual attention in professional airline pilots with a modified coefficient k: A full flight simulator study. *ACM Symposium on Eye Tracking Research and Applications*, 1–6.
- Lutz, M. (2013). *Learning python: Powerful object-oriented programming*. "O'Reilly Media, Inc."
- May, J. G., Kennedy, R. S., Williams, M. C., Dunlap, W. P., & Brannan, J. R. (1990). Eye movement indices of mental workload. *Acta psychologica*, *75*(1), 75–89.

- Moore, K., & Gugerty, L. (2010). Development of a novel measure of situation awareness: The case for eye movement analysis. *Proceedings of the human factors and ergonomics society annual meeting*, 54(19), 1650–1654.
- Mouratille, D., Amadiou, F., & Matton, N. (2022). A meta-analysis on air traffic controllers selection: Cognitive and non-cognitive predictors. *Journal of Vocational Behavior*, 138, 103769.
- OGAMA. (2024a). *Documentation*. Retrieved March 13, 2024, from <http://ogama.net/sites/default/files/pdf/OGAMA-DescriptionV25.pdf>
- OGAMA. (2024b). *Sources*. Retrieved March 13, 2024, from <https://github.com/avosskuehler/ogama>
- Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science & Engineering*, 9(3), 10–20. <https://doi.org/10.1109/MCSE.2007.58>
- Palma Fraga, R., Kang, Z., Crutchfield, J. M., & Mandal, S. (2021). Visual search and conflict mitigation strategies used by expert en route air traffic controllers. *Aerospace*, 8(7), 170.
- PupilLabs. (2024a). *Documentation*. Retrieved March 13, 2024, from <https://docs.pupil-labs.com/>
- PupilLabs. (2024b). *Nerf studio*. Retrieved March 13, 2024, from <https://docs.pupil-labs.com/alpha-lab/nerfs/>
- PupilLabs. (2024c). *Pupil core*. Retrieved March 13, 2024, from <https://docs.pupil-labs.com/core/software/pupil-capture/>
- PupilLabs. (2024d). *Real-time screen gaze*. Retrieved March 13, 2024, from <https://github.com/pupil-labs/real-time-screen-gaze>
- PupilLabs. (2024e). *Reference image mapper*. Retrieved March 13, 2024, from <https://docs.pupil-labs.com/enrichments/reference-image-mapper/>
- PupilLabs. (2024f). *Sources*. Retrieved March 13, 2024, from <https://github.com/pupil-labs>

- PupilLabs. (2024g). *Surface tracking*. Retrieved March 13, 2024, from <https://docs.pupil-labs.com/core/software/pupil-capture/#surface-tracking>
- PyGaze. (2024). *Sources*. Retrieved March 13, 2024, from <https://github.com/esdalmaijer/PyGazeAnalyser>
- PyTrack. (2024a). *Documentation*. Retrieved March 13, 2024, from <https://pytrack-ntu.readthedocs.io/en/latest/index.html>
- PyTrack. (2024b). *Sources*. Retrieved March 13, 2024, from <https://github.com/titoghose/PyTrack>
- Reichle, E. D., Pollatsek, A., Fisher, D. L., & Rayner, K. (1998). Toward a model of eye movement control in reading. *Psychological review*, *105*(1), 125.
- Roberts, D., Johnson, R., et al. (1996). Evolving frameworks: A pattern language for developing object-oriented frameworks. *Pattern languages of program design*, *3*, 471–486.
- Salvucci, D. D., & Goldberg, J. H. (2000). Identifying fixations and saccades in eye-tracking protocols. *Proceedings of the 2000 symposium on Eye tracking research & applications*, 71–78.
- Shafiee, M. J., Chywl, B., Li, F., & Wong, A. (2017). Fast yolo: A fast you only look once system for real-time embedded object detection in video. *arXiv preprint arXiv:1709.05943*.
- SmartEye. (2024). *Project description*. Retrieved March 13, 2024, from <https://www.smarteye.se/solutions/behavioral-research/aviation-aerospace/>
- Theeuwes, J., Belopolsky, A., & Olivers, C. N. (2009). Interactions between working memory, attention and eye movements. *Acta psychologica*, *132*(2), 106–114.
- TobiiProLab. (2024a). *Assisted mapping*. Retrieved March 13, 2024, from <https://connect.tobii.com/s/article/how-to-perform-manual-and-assisted-mapping>
- TobiiProLab. (2024b). *Product description*. Retrieved March 13, 2024, from https://go.tobii.com/tobii_pro_lab_product_description

TobiiProLab. (2024c). *User manual*. Retrieved March 13, 2024, from

https://go.tobii.com/tobii_pro_lab_user_manual

Voßkühler, A., Nordmeier, V., Kuchinke, L., & Jacobs, A. M. (2008). Ogama (open gaze and mouse analyzer): Open-source software designed to analyze eye and mouse movements in slideshow study designs. *Behavior research methods*, *40*, 1150–1162.

Wang, Y., Wang, L., Lin, S., Cong, W., Xue, J., & Ochieng, W. (2021). Effect of working experience on air traffic controller eye movement. *Engineering*, *7*(4), 488–494.

Appendix

GazeAnalysis submodule

The GAZEANALYSIS submodule contains ready-to-use algorithms to start using ARGAZE.

GazeMovementIdentifiers algorithms

- **DispersionThresholdIdentification:** implementation of the I-DT algorithm as described in Salvucci and Goldberg, 2000.
- **VelocityThresholdIdentification:** implementation of the I-VT algorithm as described in Salvucci and Goldberg, 2000.

ScanPathAnalyzers algorithms

- **Basic:** metrics about numbering, summing, averaging values related to fixations or saccades.
- **ExploreExploitRatio:** implementation of the explore/exploit ratio algorithm as described in Dehais et al., 2015. It takes into account saccades, short fixation (80-120 ms), and long fixations (240-260 ms). When this ratio increase, short fixations and saccades dominate: it reflects the search for information. When the ratio decrease, long fixations dominate: the individual is considered to exploit or process information, for example in an instrument.
- **KCoefficient:** implementation of the K-coefficient algorithm as described in Krejtz et al., 2016.
- **NearestNeighborIndex:** implementation of Nearest Neighbor Index algorithm as described in Di Nocera et al., 2006. This metrics index the level of randomness in the distribution of eye fixations. It is the ratio between the average of the observed minimum distances between fixation points and the mean random distance that would be expected if the distribution was random. A ratio equal to 1 would be a totally random distribution. According to Di Nocera et al., 2006, the index increase with mental workload, reflecting more dispersed visual patterns aiming at optimizing promptness to incoming information. This index can be used online.

AOIMatcher algorithms

- **DeviationCircleCoverage:** considers the intersection between AOI shape and a circle based on fixation deviation.
- **FocusPointInside:** considers the mean point of a fixation.

AOIScanPathAnalyzers algorithms

- **Basic:** metrics about numbering, summing, averaging values related to AOI matchings.
- **Entropy:** implementation of the entropy algorithm as described in Krejtz et al., 2014.
- **KCoefficient:** implementation of the K-Modified coefficient algorithm as described in C. A. Lounis et al., 2020.
- **LempelZivComplexity:** implementation of Lempel-Ziv complexity algorithm as described in C. Lounis et al., 2020.
- **NGram:** Implementation of N-Gram algorithm as proposed in C. Lounis et al., 2021.
- **TransitionMatrix:** Implementation of transition matrix probabilities and density algorithm as described in Krejtz et al., 2014.

GazePositionCalibrator algorithms

- **LinearRegression:** implementation of linear regression algorithm as described in Drewes et al., 2019.