

Hellenic Mediterranean University - Department of Electronic Engineering

ALPER-EU Project - K12 Agile Learning of Programming with Educational Robotics

Development of a Multisensor Framework for Objective Student Attention Assessment

Final Year Thesis Report

Integrated MEng Electronic Engineering

David Boluwatife Akoda

ΤΛ20181

Supervisors: Antonios Konstantaras (Professor)

Nikolaos Spanoudakis (Professor)

Georgios Giannakakis (Professor)

Academic Year 2025–2026

Chania, Crete





Ελληνικό Μεσογειακό Πανεπιστήμιο - Τμήμα Ηλεκτρονικών Μηχανικών

Έργο ALPER-EU - Ευέλικτη Μάθηση Προγραμματισμού με Εκπαιδευτική Ρομποτική

Ολοκληρωμένο Σύστημα και Λογισμικό Καταγραφής και Αξιολόγησης της Συγκέντρωσης Μαθητών

Διπλωματική Εργασία

Ενιαίο Μεταπτυχιακό Δίπλωμα Ηλεκτρολόγου Μηχανικού

Ντεϊβιντ Μπολουγουατιφε Ακοντα

ΤΛ20181

Επιβλέποντες: Αντώνιος Κωνσταντάρας (Καθηγητής)

Νικόλαος Σπανουδάκης (Καθηγητής)

Γεώργιος Γιαννακάκης (Καθηγητής)

Ακαδημαϊκό Έτος 2025–2026

Χανιά, Κρήτη



Table of Contents

Abstract 5

1. Introduction 6

1.1. Background and Motivation 6

1.2. Problem Definition 7

1.3. Objectives 8

1.4. Scope and Limitations 8

1.5. Thesis Structure 9

2. User Needs Analysis 9

2.1. Purpose and Context 9

2.2. Target and Object of Measurement 10

2.3. Measurable Behavioral Indicators 10

2.4. Functional Requirements 11

2.5. Non-Functional Requirements 12

2.6. Ethical Considerations 12

2.7. Summary 13

3. Design Phase of the Multimodal Engagement Assessment System 13

3.1. System Overview 13

3.2. System Architecture 14

3.3. Mapping: User Needs → Design Decisions 22

3.4. Data Design & Flow 25

3.5. Behavior & Focus Design 34

3.6. Design for Interpretability & Evaluation 43

3.7. Constraints & Ethics by Design 45

3.8. Design Phase Summary 47

4. Implementation Phase of the Multimodal Engagement Assessment System 48

4.1. Introduction 48

4.2. Development Environment and Tools 49

4.3. Software Structure and Module Organization 52

4.4. Implementation of Gaze Data Acquisition 60

4.5. Implementation of Body Tracking and Video Recording 64

4.6. Implementation of Synchronization 71
4.7. Implementation of the Graphical User Interface 75
4.8. Implementation of Data Storage and Session Organization 80
4.9. Implementation of Engagement Analysis and Reporting 84
4.10. Error Handling and Practical Robustness 89
4.11. Sensor Interface Requirements and Extensibility 93
4.12. Implementation Phase Summary 94
5. Results & Evaluation 95
5.1. Overview 95
5.2. System Startup and Session Initialization 96
5.3. Live Engagement Monitoring During Recording 98
5.4. Post Recording Verification and Output Files 102
5.5. Real Time Body Tracking and Engagement Classification 105
5.6. Post Session Engagement Analysis 110
5.7. Excel Teacher Report 115
5.8. System Stability and Performance 121
5.9. Discussion of Results 121
5.10. System Limitations and Directions for Improvement 123
6. Conclusions & Future Work 124
6.1. Conclusion 124
6.2. Future Work 125
7. Appendices 126
7.1. Appendix A: Installation Manual 126
7.2. Appendix B: User Manual 130
7.3. Appendix C: Hardware Substitution Guide 133
References 137

Abstract

This thesis presents the design and implementation of the ALPER-EU Multimodal Engagement Assessment System, a teacher-oriented platform for the objective measurement of student behavioral engagement during educational robotics and programming activities. The work is conducted within the ALPER-EU research project (Agile Learning of Programming with Educational Robotics) at the Hellenic Mediterranean University and addresses a fundamental limitation of unimodal monitoring approaches: neither gaze nor body signals alone can reliably distinguish task-related off-screen activity, such as hands-on robot interaction, from genuine disengagement (Bosch et al., 2015; Monkaresi et al., 2017).

The system integrates two complementary sensing modalities: screen-based gaze tracking using the Tobii Eye Tracker 4C (Tobii AB, 2021) and body orientation and depth-based motion analysis using the Intel RealSense D455 depth camera (Intel Corporation, 2023). Body pose estimation is performed using MediaPipe with a lightweight configuration to sustain real-time operation at 30 frames per second. The two sensing pipelines are synchronized through a two-stage mechanism. A gate-based shared start timestamp ensures that both streams share a common temporal origin, while nearest-neighbour temporal alignment using pandas merge_asof with a 50 millisecond tolerance window aligns individual samples during post-session analysis (Zaletelj & Košir, 2017).

Engagement is quantified through a transparent, rule based Focus Index, defined as a continuous score ranging from 0 to 100, which measures screen directed behavioral engagement. This metric reflects the degree to which a student's observable actions are oriented toward the screen based workstation. It does not represent overall task engagement or cognitive state. As a result, during hands on robot interaction phases, reduced scores are expected, since gaze and body orientation are appropriately directed away from the screen.

The score is computed as a weighted linear combination of three behavioral signals: eye gaze direction with weight 0.50, body orientation with weight 0.30, and student to camera distance with weight 0.20. This formulation follows established approaches in multimodal engagement classification (Booth et al., 2023; Dewan et al., 2019; Pabba & Kumar, 2022). Per frame scores are aggregated into one second intervals and smoothed using a three second centered rolling average to reduce sensitivity to transient attention shifts (Alkabbany et al., 2023; Sümer et al., 2021).

The resulting values are interpreted across three zones: High at or above 80, Moderate between 50 and 79, and Low below 50. A binary engagement verdict is additionally computed using a conservative three signal disengagement rule, where a student is classified as disengaged only when all three behavioral signals fail simultaneously. This design ensures that off screen robot interaction is not misclassified as disengagement (Booth et al., 2023; Raca & Dillenbourg, 2013).

The system is implemented as a modular Python application comprising four primary modules. `MainApp.py` manages session orchestration and the graphical user interface, `GazeAppAlpha.py` handles gaze acquisition via ZeroMQ, `body-tracker.py` performs RealSense body tracking and synchronized video recording, and `engagement_analysis.py` implements the post-session analysis pipeline. Each session produces structured CSV output files, a synchronized XVID-encoded video recording, session metadata, and a post-session analysis folder containing engagement timelines, gaze heatmaps, disengagement event tables, and a teacher-facing Excel report (Goldberg et al., 2021; Prieto et al., 2016). The graphical user interface, implemented in Tkinter, supports single-button session control and requires no technical training to operate (Bidwell J. & Fuchs H., 2011; Lundh, 1999).

The system was preliminarily evaluated in an educational robotics session conducted within the ALPER-EU project at the Hellenic Mediterranean University. Qualitative comparison of the computed engagement profiles with teacher observations indicated that the system produced trajectories consistent with observed classroom behavior. The three-signal disengagement rule successfully avoided false classifications during hands-on robot interaction phases. Formal quantitative validation against observer-annotated labels is identified as a priority for the evaluation phase (D'Mello et al., 2017; Sümer et al., 2021).

Overall, the developed system establishes a transparent, reproducible, and deployable foundation for sensor-based behavioral engagement assessment in educational robotics environments. It demonstrates that low-cost commercially available hardware can be combined with open-source software to produce interpretable, teacher-facing engagement metrics suitable for real classroom use (Alkabbany et al., 2023).

1. Introduction

1.1. Background and Motivation

Student engagement is widely recognized as one of the most influential determinants of learning outcomes in formal education (Fredricks et al., 2004). In classroom environments, sustained attention, active participation, and goal directed behavior are consistently associated with higher academic achievement and deeper conceptual understanding. Teachers rely on their ability to observe and interpret student behavior in real time to adjust instruction, identify struggling learners, and maintain productive learning conditions. In practice, however, continuous and objective monitoring of individual engagement is extremely difficult to achieve, particularly in dynamic settings where students divide their attention between multiple learning modalities.

This challenge is especially pronounced in educational robotics and programming environments, such as those found in Agile Learning sessions, where students regularly

alternate between screen based coding tasks and hands on interaction with physical robots. During such activities, a teacher must simultaneously manage instructional flow, support students at different stages of progress, and interpret whether a student who has turned away from the screen is productively engaged with a robot or has become distracted. Unaided human observation cannot reliably distinguish between these conditions at scale, making objective behavioral data an important supplement to teacher judgment (Goldberg et al., 2021; Raca & Dillenbourg, 2013).

Traditional engagement assessment methods, including pen and paper observation instruments, self report questionnaires, and manual video coding, offer some degree of objectivity but are limited in temporal resolution, require significant post session effort, and cannot support real time awareness during a lesson (Dewan et al., 2019; D'Mello et al., 2017). Recent advances in computer vision, eye tracking, and depth sensing have enabled automated and non intrusive behavioral monitoring, capable of producing objective and reproducible engagement indicators without disrupting the learning process (Booth et al., 2023; Sümer et al., 2021).

Within this context, this thesis contributes to the field of multimodal learning analytics by designing and implementing a sensor based engagement assessment system as part of the ALPER-EU project at the Hellenic Mediterranean University. The system targets the behavioral dynamics of educational robotics classrooms and produces interpretable, teacher facing outputs that do not require technical expertise.

1.2. Problem Definition

A fundamental limitation of existing unimodal engagement monitoring approaches is their inability to distinguish task related off screen activity from genuine disengagement. Gaze based systems can reliably detect when a student looks away from a screen but cannot determine whether this shift reflects meaningful interaction, such as working with a robot, or simple distraction (Bosch et al., 2015; Monkaresi et al., 2017). Body tracking systems provide complementary information about posture, movement, and orientation but lack the specificity required to characterize visual attention independently (Araya & Sossa-Rivera, 2021; Zaletelj & Košir, 2017).

In educational robotics settings, this ambiguity has direct consequences. Students are expected to look away from the screen as part of the task. A system that misclassifies robot interaction as disengagement would produce misleading engagement profiles, reducing its usefulness and undermining teacher trust (Goldberg et al., 2021; Prieto et al., 2016).

In addition, any system intended for classroom deployment must be minimally intrusive, simple to operate, and capable of running reliably on standard hardware without requiring specialized setup. Systems that impose additional cognitive burden on teachers or produce outputs that are difficult to interpret are unlikely to be adopted in practice

regardless of their technical performance (Alkabbany et al., 2023; Bidwell J. & Fuchs H., 2011).

1.3. Objectives

The primary objective of this thesis is to design, implement, and preliminarily evaluate a multimodal engagement assessment system that addresses the limitations identified above. The system integrates gaze tracking and depth based body tracking to provide context aware interpretation of student behavior and produces structured outputs suitable for both classroom use and research analysis.

The specific objectives are:

- To design a modular software architecture that integrates the Tobii Eye Tracker 4C and the Intel RealSense D455 within a unified Python application
- To implement a two stage temporal synchronization mechanism ensuring millisecond level correspondence between gaze and body data streams
- To define and implement a transparent Focus Index that quantifies screen directed behavioral engagement using gaze direction, body orientation, and student proximity
- To implement a conservative three signal disengagement rule that avoids false disengagement during robot interaction
- To develop a minimal graphical user interface enabling single action session control without technical intervention
- To produce structured outputs including CSV files, synchronized video recordings, engagement timelines, gaze heatmaps, disengagement tables, and a teacher facing Excel report
- To conduct a preliminary evaluation within an ALPER-EU session and compare computed engagement profiles with teacher observations

1.4. Scope and Limitations

The system is designed for single student workstation monitoring within structured Agile Learning sessions. It focuses exclusively on observable behavioral indicators of engagement, including visual attention, body orientation, and physical proximity, and does not attempt to assess cognitive or emotional states. This reflects a deliberate design choice prioritizing transparency, reproducibility, and classroom deployability (Booth et al., 2023; Perrotta & Selwyn, 2020).

The Focus Index specifically measures screen directed behavioral engagement. It does not capture full task engagement in mixed modality environments. During robot interaction phases, reduced scores are expected since gaze and body orientation are directed away from the screen. This is an inherent limitation of gaze based behavioral

proxies and is addressed through the three signal disengagement rule, which prevents such periods from being misclassified as disengagement (Raca & Dillenbourg, 2013).

Multi student tracking, group interaction analysis, and unrestricted classroom movement are outside the scope of this work. Formal quantitative validation of the Focus Index weights using annotated ground truth data is identified as future work, as the current weights are based on prior literature and have not yet been calibrated for the ALPER-EU context (D'Mello et al., 2017; Sümer et al., 2021).

1.5. Thesis Structure

The remainder of this thesis is organized as follows. Chapter 2 presents the User Needs Analysis, identifying the requirements that guided the system design. Chapter 3 describes the Design Phase, including system architecture, synchronization strategy, data flow, behavioral inference model, and ethical considerations. Chapter 4 presents the Implementation Phase, detailing the software structure and module organization. Chapter 5 presents preliminary results and system validation. Chapter 6 concludes the thesis and outlines directions for future work.

2. User Needs Analysis

2.1. Purpose and Context

This chapter identifies and analyzes the needs of the primary users of the ALPER-EU Multimodal Engagement Assessment System and defines the requirements that the system must satisfy. The purpose of this analysis is to ensure that all subsequent design and implementation decisions are grounded in clearly documented user needs, maintaining a traceable connection between classroom requirements and the technical choices made throughout the project.

The system is developed within the context of Agile Learning sessions that involve both programming and hands on educational robotics activities. In these settings, students regularly alternate between screen based coding and interaction with a physical robot, making continuous teacher observation of individual engagement difficult and inherently subjective. Objective, sensor based monitoring is therefore introduced not as a replacement for teacher judgment, but as a complementary source of behavioral evidence that teachers can interpret alongside their own observations (Fredricks et al., 2004; Goldberg et al., 2021; Raca & Dillenbourg, 2013).

The requirements identified in this chapter directly inform the system architecture, behavioral inference model, storage design, and interface described in subsequent chapters.

2.2. Target and Object of Measurement

The primary users of the system are teachers and educational researchers facilitating Agile Learning activities that involve programming and educational robotics. These users interact with the system through a graphical interface to initiate and terminate recording sessions, monitor live data during a session, and review engagement outputs after the session ends. They are not expected to have technical expertise in sensor systems, computer vision, or data analysis.

The object of measurement is individual student behavioral engagement at a single workstation during a structured learning session. The system observes how a student interacts with both digital and physical learning elements by capturing the distribution of visual attention between the computer screen and the robot, as well as physical posture and proximity at the workstation. This is achieved through synchronized gaze and body motion data. The system is designed around a one student per station configuration, reflecting common classroom setups and simplifying behavioral interpretation (Booth et al., 2023).

2.3. Measurable Behavioral Indicators

The system measures student engagement through two synchronized sensor streams, each providing complementary behavioral information.

Gaze Direction (Tobii Eye Tracker 4C): The Tobii Eye Tracker 4C captures real time gaze coordinates at sampling rates of up to 90 Hz in standalone mode and approximately 30 Hz during synchronized dual sensor recording. The primary signal derived from this stream is a binary on screen or off screen classification. A sample is marked as on screen when the gaze point falls within the display boundary, and off screen otherwise. Gaze based signals are consistently identified as strong indicators of visual attention in classroom environments (Goldberg et al., 2021; Sümer et al., 2021; Whitehill et al., 2014).

Off screen events are not automatically interpreted as disengagement. In educational robotics contexts, students are expected to look away from the screen during robot interaction. The system therefore combines gaze direction with body orientation to ensure context aware interpretation (Bosch et al., 2015; Monkaresi et al., 2017).

Body Orientation and Proximity (Intel RealSense D455): The Intel RealSense D455 provides synchronized RGB and depth streams at 30 frames per second. Body pose

estimation is performed using MediaPipe, extracting upper body landmarks including the nose and both shoulders.

Two behavioral signals are derived:

- **Facing Forward** indicates whether the student is oriented toward the screen, based on relative shoulder position and depth
- **Distance OK** confirms that the student is within the operational range of the camera, ensuring measurement validity and presence at the workstation

Body orientation acts as a complementary signal to gaze, improving interpretation of attention and engagement (Araya & Sossa-Rivera, 2021; Chen et al., 2025; Zaletelj & Košir, 2017).

Hand Activity (Intel RealSense D455): Hand landmark detection produces a Fingers Detected flag and a Hand Count value per frame. These signals are stored separately and provide an indicator of interaction with the robot or keyboard. While not currently included in the Focus Index computation, they support post session analysis and future extensions of the system.

Combined Engagement Representation: The three primary signals, gaze direction, body orientation, and distance, are combined to compute the Focus Index, a continuous score from 0 to 100 representing screen directed behavioral engagement (Booth et al., 2023; Dewan et al., 2019; Pabba & Kumar, 2022). The score is interpreted across three zones: High at or above 80, Moderate between 50 and 79, and Low below 50.

A separate binary engagement verdict is computed using a conservative three signal rule, ensuring that off screen activity during robot interaction is not misclassified as disengagement. The full formulation is presented in Chapter 3.

2.4. Functional Requirements

The following requirements were identified based on user needs and classroom context. Each requirement is assigned an identifier for traceability.

- **UR-01 Simple interaction during class**
  The system must be activated through a single action and operate without manual intervention during recording
- **UR-02 Objective engagement assessment**
  The system must produce quantitative and interpretable engagement indicators based on observable behavior

- **UR-03 Context aware interpretation**
  The system must distinguish between disengagement and task related off screen activity
- **UR-04 Temporal synchronization**
  Sensor data streams must be aligned at millisecond resolution
- **UR-05 Session validation capability**
  The system must provide synchronized video for visual verification
- **UR-06 Structured and reproducible data**
  All data must be stored in organized formats supporting analysis and comparison
- **UR-07 Minimal classroom disruption**
  The system must operate passively without affecting natural student behavior

2.5. Non-Functional Requirements

- **Temporal accuracy**
  Millisecond level synchronization with a 50 millisecond alignment tolerance
- **Real time performance**
  Stable operation at 30 frames per second
- **Usability**
  Simple interface requiring no technical expertise
- **Privacy and data protection**
  Compliance with institutional policies and GDPR requirements
- **Reproducibility**
  Use of open and structured data formats
- **Deployability**
  Operation on standard hardware without specialized setup

2.6. Ethical Considerations

The system is designed for educational environments involving minors, and ethical considerations are treated as core design requirements. The system does not attempt to infer cognitive or emotional states. Engagement is defined strictly in terms of observable behavior, and outputs are intended to support teacher reflection rather than evaluation or labeling (Holmes et al., 2022; Perrotta & Selwyn, 2020).

All monitoring is passive and non intrusive. The system does not require wearable devices, does not interact with students, and does not provide real time feedback. Teachers retain full control over session activation and data collection.

Transparency is supported through synchronized video recordings, allowing validation of system outputs against observed classroom behavior and ensuring that the system remains interpretable and trustworthy (Goldberg et al., 2021; Raca & Dillenbourg, 2013).

2.7. Summary

This chapter identified seven core user requirements spanning functional and non functional dimensions, along with ethical constraints appropriate for classroom deployment. These requirements establish the foundation for all subsequent design and implementation decisions. Their traceability to system architecture and components is presented in Chapter 3.

3. Design Phase of the Multimodal Engagement Assessment System

3.1. System Overview

The ALPER-EU Multimodal Engagement Assessment System is a teacher-oriented application designed to capture and summarize student engagement during programming and educational robotics activities. It is intended for classroom settings where students frequently move between screen-based coding and hands-on interaction with a robot, making continuous observation by the teacher difficult and often subjective.

To address this challenge, the system combines two complementary sensing modalities: screen-based gaze tracking using the Tobii Eye Tracker 4C (Tobii AB, 2021) and body orientation and motion monitoring using the Intel RealSense D455 depth camera (Intel Corporation, 2023). By integrating information from these two sources, the system can provide a more contextual interpretation of attention shifts. For example, looking away from the screen can be interpreted differently depending on whether the student is actively working on the robot or disengaged from the task.

During a recording session, teachers interact with the system through a minimal interface that allows them to start and stop data collection and monitor a live preview. After the session ends, the system produces structured and easy-to-understand outputs, including an engagement timeline, gaze heatmaps, a quantitative Focus Index, and high-level behavioral states. These outputs are designed to be interpretable without requiring technical expertise.

All collected data are stored and exported in a reproducible, session-based format. This design supports later analysis, comparison across sessions, and validation through synchronized video reference, while maintaining privacy-aware handling suitable for classroom environments.

3.1.1. Design Assumptions and System Boundaries

The system is designed for use during structured Agile Learning sessions. Teachers initiate recording at the beginning of a sprint and stop it at the end of the activity. Each recording session corresponds to a single sprint and is treated as an independent unit of analysis.

It is assumed that one student operates at a single workstation equipped with a monitor, a computer, and an educational robot. The workstation is positioned within the field of view of the depth camera, and the student is expected to remain within this workspace for most of the session. Based on this assumption, the system is designed around a one-student-per-station configuration, which reflects common classroom robotics setups.

The current design focuses exclusively on observable behavioral indicators of engagement, such as visual attention, body stability, and physical interaction with learning artifacts. Engagement is defined using measurable and reproducible signals derived from multimodal sensor data, following common approaches in educational engagement research (Booth et al., 2023). The system does not attempt to directly assess internal cognitive states, learning outcomes, or emotional conditions at this stage.

More advanced interpretations, including cognitive load estimation, learning performance evaluation, or emotional state analysis, are intentionally excluded from the present design scope. These aspects are identified as potential future extensions, to be explored only after the behavioral engagement indicators and synchronization mechanisms of the current system have been validated.

Finally, group-level interaction analysis, multi-student tracking using a shared sensor setup, and unrestricted student movement across the classroom are considered outside the boundaries of the current design phase. While these scenarios are acknowledged as meaningful directions for future work, they are not addressed in the present system in order to preserve robustness, interpretability, and alignment with the intended pedagogical context.

3.2. System Architecture

The system architecture of the ALPER-EU Multimodal Engagement Assessment System follows a modular, layered design. Each major function of the system is assigned to a

separate layer, including sensing, synchronization, processing, inference, storage, and user interaction. This separation improves clarity, supports future extension, and helps the system remain robust in real classroom environments.

Figure 1 presents a high-level view of the system architecture, illustrating the main functional layers and the flow of multimodal data and user interaction (Booth et al., 2023; Raca & Dillenbourg, 2013).

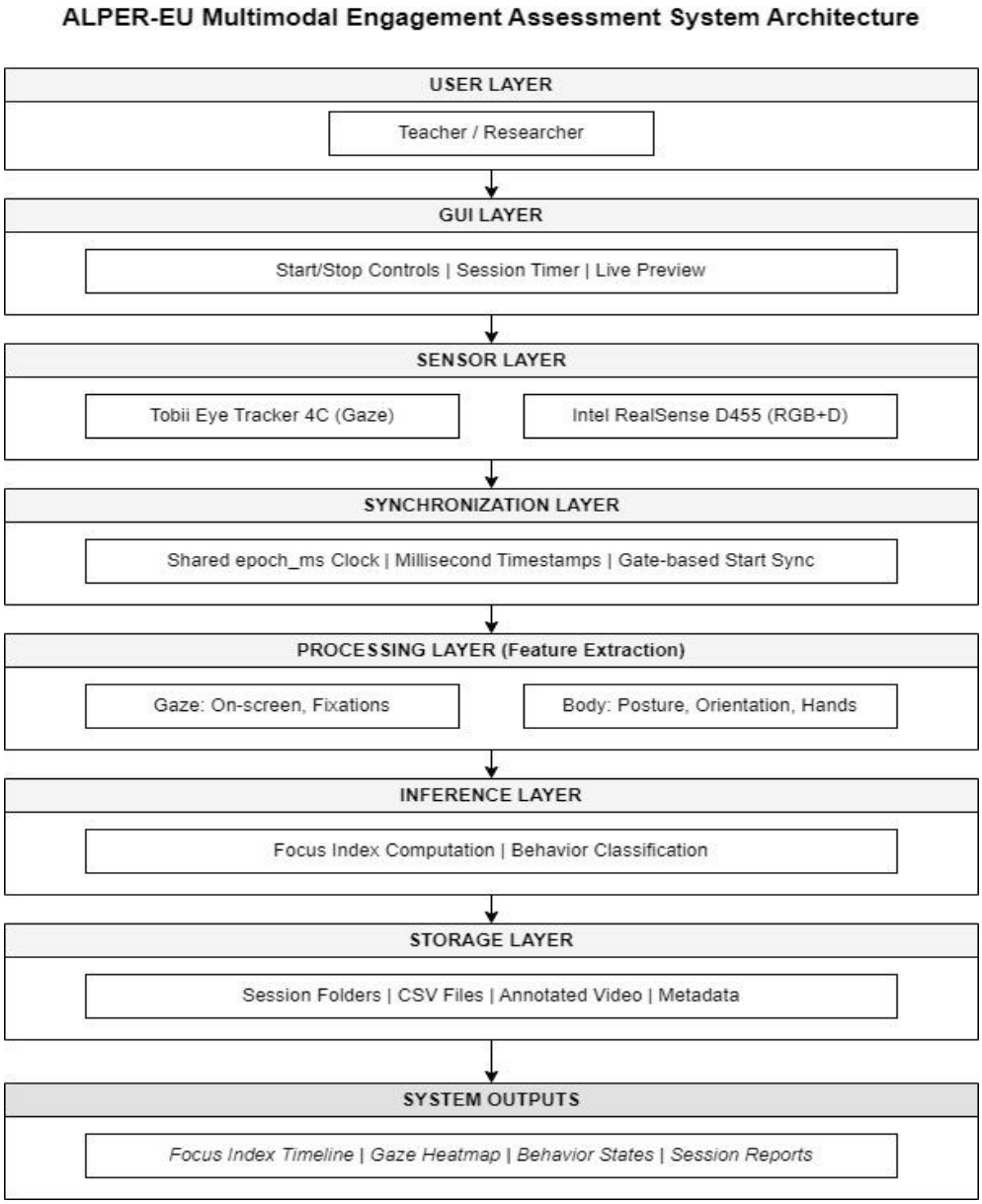


Figure 1 - Layered system architecture of the proposed multimodal engagement assessment system.

3.2.1. Sensor Layer

The Sensor Layer is responsible for collecting raw multimodal data that describe students' visual attention and physical behavior. Two complementary sensing devices are used.

Screen-based gaze data are captured using the Tobii Eye Tracker 4C, which provides real-time information about gaze position and fixation within the active display area (Tobii AB, 2021). Gaze-based signals have been shown to reliably indicate student attention and cognitive workload in learning contexts (Appel et al., 2018; Whitehill et al., 2014). In parallel, body orientation, posture, and hand movement are captured using the Intel RealSense D455 depth camera, which provides synchronized RGB and depth streams suitable for pose estimation and motion analysis (Intel Corporation, 2023). Depth-sensor-based body tracking has been applied to student attention estimation in classroom environments, demonstrating the suitability of this approach for non-intrusive monitoring (Asteriadis et al., 2013; Zaletelj & Košir, 2017).

The selection of sensing hardware was guided by three criteria: the established relevance of each modality to observable engagement signals in educational research, practical suitability for classroom deployment, and compatibility with the single-student workstation configuration described in Section 3.1. The Tobii Eye Tracker 4C was selected because it provides accurate remote gaze tracking at a fixed workstation and integrates with the system through a local TCP stream, avoiding dependency on proprietary or restricted APIs. The Intel RealSense D455 was selected because it provides synchronized RGB and depth streams within a single device, enabling both pose estimation via MediaPipe and depth-based distance measurement without requiring additional hardware.

The use of two distinct sensors is a deliberate architectural choice. By combining gaze and body information, the system can interpret off-screen behavior more accurately, distinguishing between productive actions, such as working on a robot, and disengagement. This multimodal approach addresses a known limitation of unimodal methods, where gaze or body signals alone are insufficient to reliably separate task-related activity from disengagement (Bosch et al., 2015; Monkaresi et al., 2017). By integrating both modalities, the system achieves a more context-aware and robust interpretation of student engagement.

3.2.2. Synchronization Layer

The Synchronization Layer ensures that gaze data and body-tracking data, collected from two independent sensing pipelines, are accurately aligned in time for multimodal analysis.

Because the two sensors operate independently and produce data at different rates, a unified temporal reference is required. To achieve this, all data samples are timestamped by the host Python process at the moment they are received from each sensor. Timestamps are

expressed in milliseconds since the Unix epoch, which represents the elapsed time since 1 January 1970 (UTC) and serves as a universal time reference. These timestamps are computed using the system clock via Python's `time.time_ns()` function and converted into epoch milliseconds (`epoch_ms`). This approach provides a consistent and shared time base across all recorded data and forms the foundation of the synchronization strategy. The internal hardware clocks of the sensors are not used for cross-sensor alignment, ensuring consistency across both data streams.

To ensure that both sensing pipelines begin recording simultaneously, a gate-based synchronization mechanism is used. Before each session starts, the system computes a future timestamp (`go_epoch_ms`) that defines the exact moment when data recording should begin. This value is calculated once before either sensor thread is started and is passed to both sensing processes during initialization. Each process monitors the system clock and only begins writing data when the predefined time is reached. This approach ensures that both data streams share the same temporal origin, avoiding inconsistencies caused by initialization delays or thread scheduling.

Despite sharing a common time reference, the two sensors operate at different sampling rates. The body-tracking pipeline produces data at approximately 30 frames per second, while the eye-tracking sensor can produce gaze data at up to 90 Hz in standalone mode and approximately 30 Hz during synchronized recording. As a result, samples from the two streams do not occur at identical timestamps.

To address this, temporal alignment is performed during the analysis stage using nearest-neighbor matching, a method that associates each body frame with the gaze sample closest in time. The body-tracking stream is used as the temporal reference due to its lower and more stable sampling rate. When multiple gaze samples fall within the matching window, only the sample with the smallest temporal difference is retained, ensuring a single and consistent gaze representation for each frame (Zaletelj & Košir, 2017).

A tolerance window of 50 milliseconds is applied during this process. This threshold corresponds approximately to one to two frames at 30 frames per second and provides a balance between alignment accuracy and tolerance to minor timing variations.

If no gaze sample is found within this window, a default off-screen value is assigned. Such cases are uncommon under normal conditions and typically occur during brief interruptions such as natural eye blinks or temporary tracking loss.

All sensors operate on a single host machine and share the same system clock. This eliminates the need for cross-device clock synchronization and ensures consistent application of the temporal alignment process.

Overall, the synchronization strategy combines a synchronized recording start with tolerance-based temporal alignment. This two-stage approach provides reliable and practical synchronization without requiring hardware-level clock coordination.

3.2.3. Processing Layer

The Processing Layer transforms raw sensor data into structured and interpretable features. For gaze data, this includes identifying on-screen versus off-screen events, fixation duration, and gaze stability. For body-tracking data, extracted features include joint positions, shoulder stability, hand activity, and overall body orientation.

No behavioral interpretation is performed at this stage. Instead, the layer produces normalized and time-aligned descriptors that serve as input for higher-level analysis. By separating signal processing from behavioral inference, the system maintains transparency and allows individual components to be improved independently (Booth et al., 2023; Dewan et al., 2019).

3.2.4. Inference Layer

The Inference Layer combines processed gaze and body features to derive high-level engagement representations. Using the structured, time-aligned features defined in the previous stage, the system applies rule-based behavioral logic grounded in observable signals such as gaze direction, body orientation, and distance from the workstation. The definition of these rules and their interpretation are presented in detail in Section 3.5.

The system computes a quantitative Focus Index that summarizes screen-directed behavioral engagement trends over time based on observable behavioral cues and classifies behavior into discrete states, such as focused screen interaction, robot-related activity, and distraction. The Focus Index quantifies the degree to which a student's observable actions are oriented toward the workstation, rather than attempting to directly infer task engagement or cognitive state. These representations are designed to be easily interpretable and directly grounded in observable behavior, making them suitable for classroom use.

Automated engagement detection using sensor-based signals has been demonstrated across a range of educational contexts, from individual tutoring to classroom instruction (Booth et al., 2023; Dewan et al., 2019). Machine learning-based approaches are intentionally excluded from the current design. The rule-based structure provides a transparent and explainable foundation, ensuring that engagement outcomes can be easily understood and validated. It also allows future extension toward data-driven methods once sufficient data have been collected and validated.

3.2.5. Storage Layer

The Storage Layer handles the persistent and structured storage of all data produced during a recording session. Each session is saved as a self-contained folder, named with a timestamp, containing all raw sensor outputs, session metadata, and post-session analysis results.

Per-Session Raw Output Files

Each completed recording session produces the following files:

Table 1 - Overview of Recorded Data Files, Sources, Sampling Rates, and Contents

File	Source	Rate	Contents
`tobii_gaze.csv`	Tobii Eye Tracker 4C	30 FPS (synchronized) / up to 90 FPS (standalone)	epoch_ms, normalized gaze coordinates x and y, on_screen flag
`upper_body.csv`	Intel RealSense D455	30 FPS	epoch_ms, device timestamp, Nose and Shoulder joint positions and depth distances, Distance_OK, Facing_Forward, Body_Engaged flags
`hands_data.csv`	Intel RealSense D455	30 FPS	epoch_ms, device timestamp, Fingers_Detected, Hand_Count
`realsense_color.avi`	Intel RealSense D455	30 FPS	Annotated RGB video at 1280×720, XVID codec
`session_metadata.json`	GUI	—	Student ID, task name, facilitator, session notes, recorded-at timestamp

Design Rationale

Separating the data into modality-specific files preserves the independence of each sensor stream and allows any individual file to be re-analyzed or replaced without affecting the others. Raw sensor data is kept distinct from derived engagement results, which are produced separately during the analysis phase and stored in an `analysis/` subfolder within the same session directory (Alkabbany et al., 2023; Raca & Dillenbourg, 2013).

The session-based folder structure supports reproducibility, comparison across sessions, and validation through synchronized video reference. All files within a session share the same `epoch_ms` timestamp reference, ensuring that any file can be cross-referenced with any other.

Note on Original Design Specification

Early design documentation described three body-tracking CSV files, including a separate file for remaining lower-body joints. Following a review of the system requirements with the project supervisor, the scope of body-tracking data collection was refined to focus on upper-body joints and hand activity, which are the behaviorally relevant signals for the engagement model. The remaining-joints file was therefore excluded from the final design.

3.2.6. Graphical User Interface (GUI) Layer

The GUI Layer provides the main interaction point for teachers. It includes simple Start and Stop controls, a session timer, and a live preview showing synchronized gaze and body-tracking overlays.

The interface is intentionally minimal and designed for ease of use without technical training. By triggering all underlying system layers simultaneously and providing clear feedback on recording status and data export, the GUI supports transparency and trust during classroom operation (Bidwell J. & Fuchs H., 2011; Goldberg et al., 2021).

3.2.7. Implementation Platform and Technology Stack

The system is implemented entirely in Python (version 3.10 or newer), selected for its broad ecosystem of scientific and computer vision libraries, native support on both the Tobii and RealSense SDKs, and suitability for rapid research prototyping.

Table 2 - Summary of the main software components used across the system layers.

Component	Role in the System
Python 3.10+	Primary implementation language across all layers
Tkinter	Graphical user interface — session control, live monitoring, analysis views
pyrealsense2	Python wrapper for the Intel RealSense SDK; provides access to depth and color streams from the D455
MediaPipe	Body pose and hand landmark detection from RealSense color frames; lightweight pose model (`complexity=0`) used to maintain 30 FPS throughput
ZeroMQ (pyzmq)	Non-blocking SUB socket interface to the TobiiStream service, which publishes gaze data over a local TCP connection on tcp://127.0.0.1:5556
OpenCV	Camera frame capture, annotated video recording (XVID/AVI), and live preview rendering
pandas	CSV loading, timestamp-based stream merging (`merge_asof`), and per-second aggregation during analysis
NumPy	Numerical operations across all processing and inference stages
Matplotlib	Generation of all analysis plots — engagement timeline, gaze heatmap, signal summary, and score chart
SciPy	Gaussian smoothing for gaze density heatmaps
openpyxl	Excel report generation for teacher-facing session summaries
Pillow	Inline image display within the Tkinter GUI

Key Design Decisions

The choice of ZeroMQ for Tobii data reception warrants specific explanation. The Tobii Eye Tracker 4C does not provide a direct Python API in the configuration used by this system. Instead, gaze data are published by the TobiiStream local service over a TCP socket. ZeroMQ provides a clean and non-blocking subscriber interface to this stream, allowing continuous data reception without interrupting the main application flow. A 500 millisecond silence threshold is used to detect potential device disconnection, ensuring robustness without blocking the main thread.

MediaPipe's lightest pose model (`model_complexity = 0`) was selected deliberately to maintain real time performance within the 33 millisecond frame budget required for 30 frames per second operation. The shoulder and nose landmarks provided by this model are sufficient to compute the distance and orientation signals used in the engagement model, making more complex models unnecessary (Asteriadis et al., 2013; Zaletelj & Košir, 2017).

Video is recorded in XVID format within an AVI container. This choice was made deliberately to ensure reliable and portable video recording across all target environments.

OpenCV's video writing interface supports XVID encoding natively, allowing video capture without requiring additional software installation or external dependencies.

In contrast, recording in MP4 format using H.264 encoding would require OpenCV to be compiled with FFmpeg support, which is not available by default on Windows systems and introduces an external dependency that cannot be consistently guaranteed across laboratory setups. Avoiding such dependencies ensures that the system remains easy to deploy and operate in real classroom environments.

The XVID and AVI combination therefore provides a self contained and robust recording solution while maintaining compatibility, simplicity, and reproducibility. It also produces manageable file sizes at a resolution of 1280 × 720 and a frame rate of 30 frames per second.

3.3. Mapping: User Needs → Design Decisions

This section explains how the user needs identified during the requirements analysis are reflected in the design of the system. The purpose of this mapping is to show a clear connection between what teachers need in the classroom and the design decisions made during the Design Phase.

By linking each user need to specific design decisions and system layers, the rationale behind the architecture becomes clear and traceable. This supports transparency, interpretability, and future evaluation of the system.

3.3.1. Mapping Overview

The primary users of the system are teachers facilitating Agile Learning activities that involve programming and educational robotics. Based on the requirements analysis, the main user needs focus on simple operation, objective assessment of student engagement, context-aware interpretation of behavior, and the ability to review session results after the activity.

This mapping focuses only on core user needs that directly influence system architecture and interaction design. Peripheral requirements and implementation-specific details are intentionally excluded at this stage. Table 3 summarizes how the identified user needs are addressed through concrete design decisions and corresponding system layers.

3.3.2. User Needs to Design Decisions Mapping

Table 3 - Traceability Between User Needs, Design Decisions, and System Layers

ID	User Need/Requirements	Functional (F)/Non Functional (NF)	Design Decision	System Layer(s)	Design Rationale / Outcome
UR-01	Simple and minimal interaction during class	NF	Single Start/Stop control with session timer	*GUI Layer*	Reduces teacher cognitive load and allows use without technical training
UR-02	Objective assessment of student engagement	F	Rule-based inference of Focus Index and behavioral states	Inference Layer	Provides transparent and interpretable engagement indicators based on observable behavior
UR-03	Context-aware interpretation of off-screen behavior	F	Combination of gaze tracking and body orientation data	Sensor & Processing Layers	Distinguishes disengagement from productive robot-related activity
UR-04	Precise temporal alignment of multimodal data	F	Millisecond-resolution timestamps with gate-based synchronized start and nearest-neighbour temporal alignment	Synchronization Layer	Ensures gaze and body data are analyzed within a unified temporal context
UR-05	Ability to review and validate session results	F	Export of synchronized video alongside sensor data	Storage Layer	Allows visual verification and qualitative validation of inferred engagement

ID	User Need/Requirements	Functional (F)/Non Functional (NF)	Design Decision	System Layer(s)	Design Rationale / Outcome
UR-06	Reproducible and analyzable session data	NF	Structured export of CSV files per data modality	Storage Layer	Supports post-session analysis, comparison across sessions, and research reproducibility
UR-07	Minimal classroom disruption	NF	Automated data acquisition after session start	System-wide	Allows teachers to focus on teaching rather than system operation

To further strengthen traceability between system requirements and implementation, the following table maps each requirement to the specific system component responsible for satisfying it. This provides a clear link between the defined system needs and their concrete realization within the architecture.

Table 4 - Traceability from User Needs to Satisfying System Components

Requirement ID	User Need	Satisfying Component
UR-01	Simple and minimal interaction during class	GUI layer — single Start/Stop button with session timer (`MainApp.py`)
UR-02	Objective assessment of student engagement	Inference layer — rule-based Focus Index and behavioural state classification (`engagement_analysis.py`)
UR-03	Context-aware interpretation of off-screen behaviour	Sensor and Processing layers — combined gaze tracking and body orientation (`GazeAppAlpha.py`, `body-tracker.py`)
UR-04	Precise temporal alignment of multimodal data	Synchronisation layer — go_epoch_ms gate-based start and merge_asof nearest-neighbour alignment with 50 ms tolerance (`MainApp.py`)
UR-05	Ability to review and validate session results	Storage layer — per-session synchronised video recording (realsense_color.avi, XVID encoding)
UR-06	Reproducible and analysable session data	Storage layer — structured CSV export per modality (`tobii_gaze.csv`, `upper_body.csv`, `hands_data.csv`)
UR-07	Minimal classroom disruption	System-wide — single-button automated data acquisition; no manual intervention required during session (`MainApp.py`)

3.3.3. Traceability and Design Consistency

The mapping presented above shows that each major design decision is directly motivated by a documented user need, while the requirements traceability table links these needs to specific system components. Together, these mappings ensure that the system architecture, data flow, and implementation are not arbitrary, but instead aligned with the pedagogical context and practical constraints of classroom use (Prieto et al., 2016).

In addition, this traceability provides a clear foundation for subsequent design elements, including data flow modeling, behavioral interpretation, and evaluation readiness. By maintaining this alignment throughout the Design Phase, the system design remains consistent, transparent, and easy to justify.

3.4. Data Design & Flow

This section describes how data move through the ALPER-EU Multimodal Engagement Assessment System during a recording session. The aim is to explain how multimodal data

are collected, synchronized, processed, and stored, from the moment the teacher starts a session until the final outputs are produced.

The description focuses on design choices, data organization, and interpretability, and intentionally avoids implementation-specific details. The overall data flow is designed to ensure temporal consistency, modular processing, and reproducibility.

3.4.1. Recording Session Initialization

Each recording session is explicitly started by the teacher through the graphical user interface. A single Start action activates all sensing components at the same time, including gaze tracking and body-tracking acquisition. This action establishes a common session reference that is used across all system components for timestamp alignment and data association.

Each session is stored in a dedicated folder created automatically at the moment recording begins. The folder name follows the pattern {mode}_{YYYYMMDD_HHMMSS}, where `mode` identifies the sensor configuration used (for example, `both` for synchronized dual-sensor recording, `eye` for gaze only, or `body` for body tracking only), and the timestamp reflects the local system time at session start. For example, a synchronized session started on 18 March 2026 at 14:30:22 would produce the folder recordings/both_20260318_143022/. All output files for that session are stored within this folder, and the folder name serves as a unique session identifier throughout the analysis pipeline.

Treating each session as a clearly defined unit supports repeatability and allows recordings to be analyzed independently. This structured organization also ensures that all data, metadata, and derived outputs remain consistently associated with a single session. After the session begins, data acquisition runs automatically, allowing the teacher to focus on the learning activity without further system interaction.

3.4.2. Multimodal Data Acquisition

During an active session, the system continuously collects multimodal data describing student visual attention and physical behavior. Gaze data indicate where students are looking relative to the screen, while body-tracking data describe posture, orientation, and upper-body movement within the workspace.

The two sensing pipelines operate independently and may generate data at different rates. To ensure consistent temporal alignment, all data samples are timestamped by the host Python process at the moment each sample is received from the sensor. Timestamps are

expressed in milliseconds since the Unix epoch, computed using the system clock via Python's `time.time_ns()` function. This approach establishes a unified and consistent time reference across both data streams.

Importantly, the internal hardware clocks of the sensors are not used for cross-sensor synchronization. Although the RealSense device timestamp is recorded in the body-tracking data as a secondary reference, all synchronization operations rely exclusively on the host-assigned epoch timestamp. This design choice avoids inconsistencies caused by independent device clocks and ensures reliable alignment between modalities.

Using both gaze and body data is a deliberate design choice. While gaze alone can indicate visual attention, it cannot reliably distinguish between disengagement and task-related off-screen activity. Body-tracking data provide the necessary contextual information for this distinction, enabling a more accurate interpretation of student behavior (Bosch et al., 2015; Monkaresi et al., 2017).

3.4.3. Temporal Synchronization and Alignment

Temporal alignment between the two sensing pipelines is achieved through a two-stage synchronization design that separates recording time coordination from post-session data alignment (Booth et al., 2023; Zaletelj & Košir, 2017).

In the first stage, a shared gate timestamp is established before the recording session begins. The gate value is computed as the current Unix epoch time in milliseconds plus a predefined countdown duration, and is calculated exactly once before either sensor thread is started. This value is passed to both the gaze and body-tracking engines prior to data acquisition. Each engine monitors the host system clock and does not write any data until the gate timestamp is reached. This mechanism guarantees that both data streams share a common temporal origin, regardless of differences in sensor initialization time or thread scheduling latency.

In the second stage, the two data streams are aligned at the sample level during post-session analysis. The body-tracking stream, sampled at approximately 30 frames per second, is used as the temporal anchor due to its lower and more stable sampling rate. For each body frame, the gaze sample with the closest timestamp is selected using nearest-neighbor temporal matching. When multiple gaze samples fall within the matching window, only the sample with the smallest temporal difference is retained, ensuring a single and consistent gaze representation for each body frame.

A tolerance window of 50 milliseconds is applied during this process. This threshold corresponds approximately to one to two frames at 30 frames per second and provides a

practical balance between alignment accuracy and tolerance to sampling jitter or sensor latency.

If no gaze sample falls within the specified tolerance window, the corresponding gaze values are assigned a conservative off-screen default. Gaps in gaze data occur when the eye tracker temporarily loses visibility of the student's pupils, most commonly during natural blinking or rapid head movements. Under normal seated conditions, such gaps lasting more than 50 milliseconds are infrequent.

This design cleanly separates the concerns of real-time recording and temporal alignment. During recording, both sensing pipelines operate independently at their native sampling rates without introducing synchronization overhead. Alignment is instead performed during the analysis phase, where it can be applied consistently and reproducibly to the stored data.

3.4.4. Feature Extraction and Data Structuring

After synchronization, raw sensor data are converted into structured, time-aligned features. For gaze data, these features describe on-screen and off-screen attention, fixation behavior, and gaze stability. For body-tracking data, extracted features include joint positions, posture stability, hand activity, and overall body orientation.

The extracted features are stored in modality-specific files and defined as follows.

Table 5 - Gaze features (tobii_gaze.csv)

Feature	Type	Definition
`epoch_ms`	Integer	Unix timestamp in milliseconds assigned at sample arrival
`x`, `y`	Float	Gaze point coordinates in screen pixels (reference resolution: 1920 × 1200)
`on_screen`	Boolean	True if the gaze point falls within the screen boundary; False otherwise

Table 6 - Body tracking features (upper_body.csv)

Feature	Type	Definition
`epoch_ms`	Integer	Unix timestamp in milliseconds assigned at frame arrival
`device_ms`	Float	RealSense hardware timestamp, recorded as a secondary reference
`Nose_X`, `Nose_Y`	Float	Horizontal and vertical position of the nose landmark, normalized to frame dimensions
`Nose_dist`	Float	Depth distance from the camera to the nose landmark, in meters
`L_Shoulder_X/Y/dist`, `R_Shoulder_X/Y/dist`	Float	Position and depth of the left and right shoulder landmarks
`Distance_OK`	Boolean	True if the student's distance from the workstation is within the configured threshold
`Facing_Forward`	Boolean	True if shoulder rotation indicates the student is oriented toward the screen
`Body_Engaged`	Boolean	True if both Distance_OK and Facing_Forward are true simultaneously

Feature extraction is intentionally kept separate from behavioral interpretation. This separation improves transparency and allows the same processed data to be reused with different inference approaches in future iterations of the system (Booth et al., 2023; Dewan et al., 2019).

3.4.5. Behavioral Inference and Output Generation

Processed gaze and body features are combined to produce higher-level engagement representations. Using the structured features derived from both modalities, the system applies predefined and interpretable behavioral logic to compute a quantitative Focus Index and assign discrete behavioral states, such as focused screen interaction, robot-related activity, or distraction. The definition of these behavioral rules and the computation of the Focus Index are presented in detail in Section 3.5.

The Focus Index is computed from key behavioral signals, including gaze direction, body orientation, and distance from the workstation, as defined in the extracted feature set. Discrete behavioral states are derived from rule-based combinations of these signals, ensuring that engagement interpretation remains transparent and directly linked to observable behavior.

These outputs are designed to reflect engagement trends over time rather than instantaneous judgments. This approach better matches classroom conditions, where brief attention shifts and natural interruptions are common and should not disproportionately affect overall engagement assessment.

3.4.6. Data Export and Session Outputs

At the end of a recording session, all collected and derived data are stored in a structured, session-based format. Each session produces modality-specific CSV files, synchronized video recordings, and session metadata.

This organization supports post-session review, comparison across activities, and reproducibility. Separating raw data from derived engagement results also ensures clear traceability from sensor input to final outputs.

3.4.7. Data Flow Visualization

Figure 2 illustrates the runtime behaviour of the proposed multimodal engagement assessment system during a synchronized recording session. The sequence begins with the teacher initiating the session through the graphical user interface, which triggers the main application to start the recording workflow. Before data acquisition begins, the system collects session metadata, such as student identification and task information, ensuring that each recording is properly contextualized and stored.

Following initialization, the main application computes a shared synchronization timestamp, denoted as `go_epoch_ms`. This timestamp acts as a global temporal reference that aligns all independent data acquisition processes. Both the gaze tracking process (Tobii) and the body tracking process (RealSense) are initialized with this shared value and remain idle until the system clock reaches the specified threshold. This mechanism guarantees that all sensor streams begin recording in a temporally synchronized manner without requiring tight coupling between processes.

Once the synchronization condition is satisfied, the system enters a parallel execution phase, where gaze data, skeletal tracking data, and video streams are recorded concurrently. The Tobii process continuously appends gaze samples to a dedicated CSV file, while the RealSense process records upper body and hand tracking data alongside color video frames. At the same time, both processes transmit lightweight state information back to the main application, enabling real-time monitoring.

The main application periodically evaluates user engagement by aggregating signals from multiple modalities, including gaze direction, body orientation, and distance from the sensor. Engagement indicators are updated in the user interface, providing immediate feedback during the session. Importantly, a disengaged state is only detected when all relevant signals indicate absence of engagement, ensuring robustness against transient noise in individual modalities.

Upon termination of the session by the user, the main application issues stop commands to all acquisition processes. Each process then finalizes its respective outputs, closing data streams and ensuring file integrity. A verification step follows, during which the system checks the completeness and consistency of the recorded data, including row counts, approximate frame rates, and video file size. The results of this verification are then presented to the user through the interface.

Overall, this sequence highlights the system's ability to coordinate multiple asynchronous components through a shared temporal reference, enabling reliable and synchronized multimodal data collection in real time.

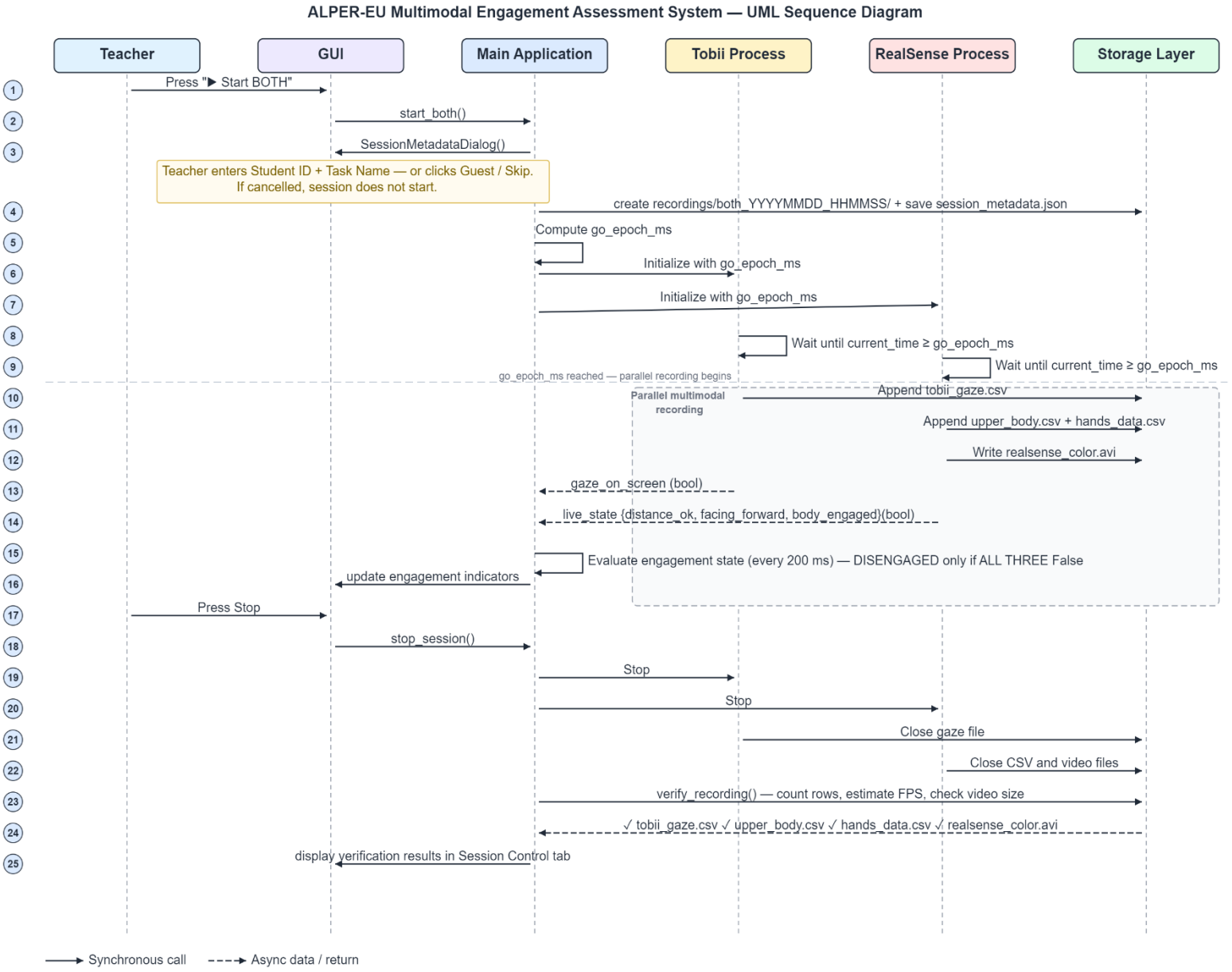


Figure 2 - UML Sequence Diagram of a Synchronized Multimodal Recording Session.

To complement the sequence diagram, Figures 3 and 4 present the system's data flow using Data Flow Diagram (DFD) notation. Figure 3 provides the context diagram (Level 0), illustrating the system boundary, external entities, and the primary data exchanges between the system and its environment. Figure 4 presents the Level 1 decomposition, offering a more detailed view of the internal processes, data stores, and the flow of multimodal data throughout the system. Together, these diagrams provide a structured and comprehensive representation of how data are captured, transformed, and propagated across the system.

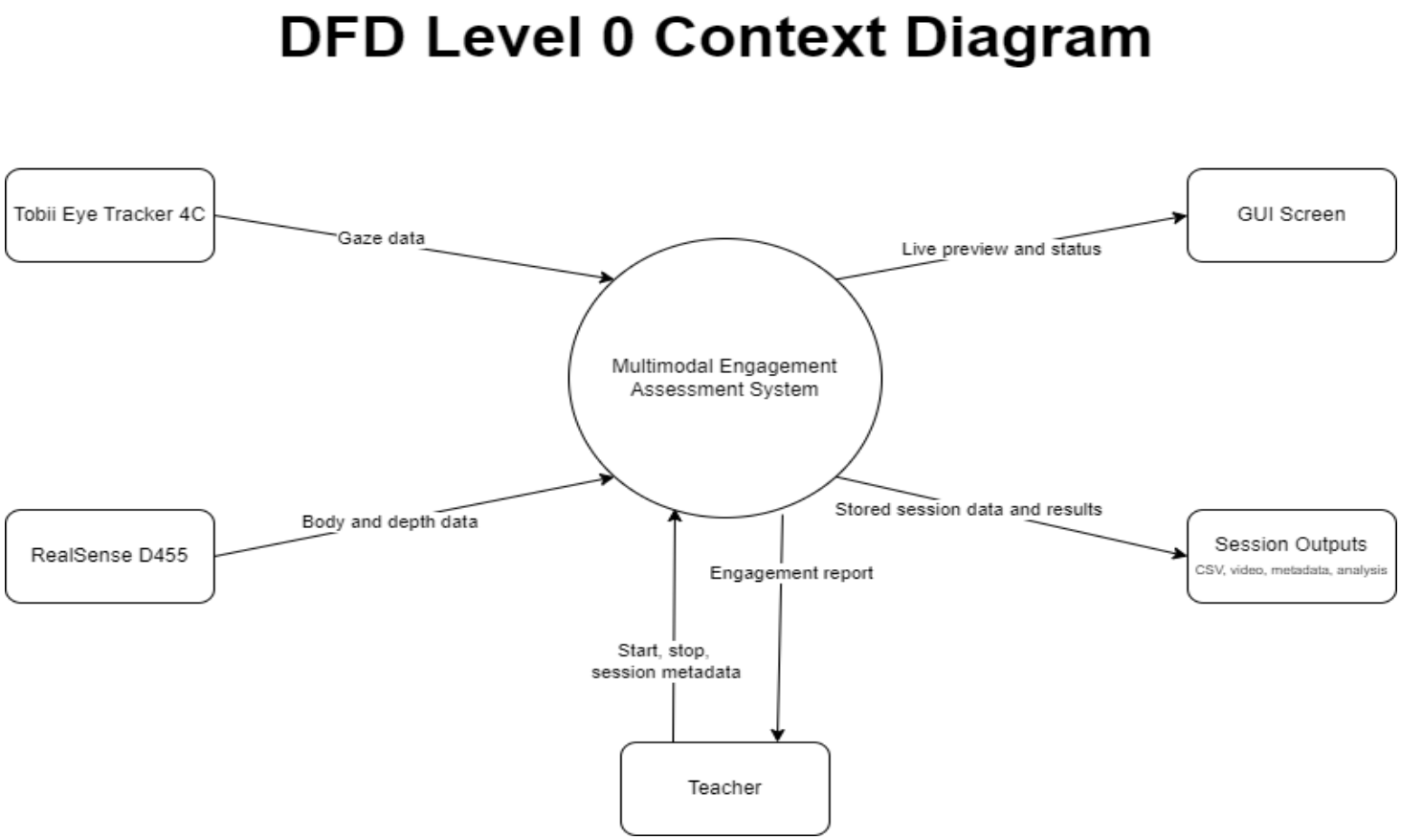


Figure 3– DFD Level 0 Context Diagram of the ALPER-EU Multimodal Engagement Assessment System.

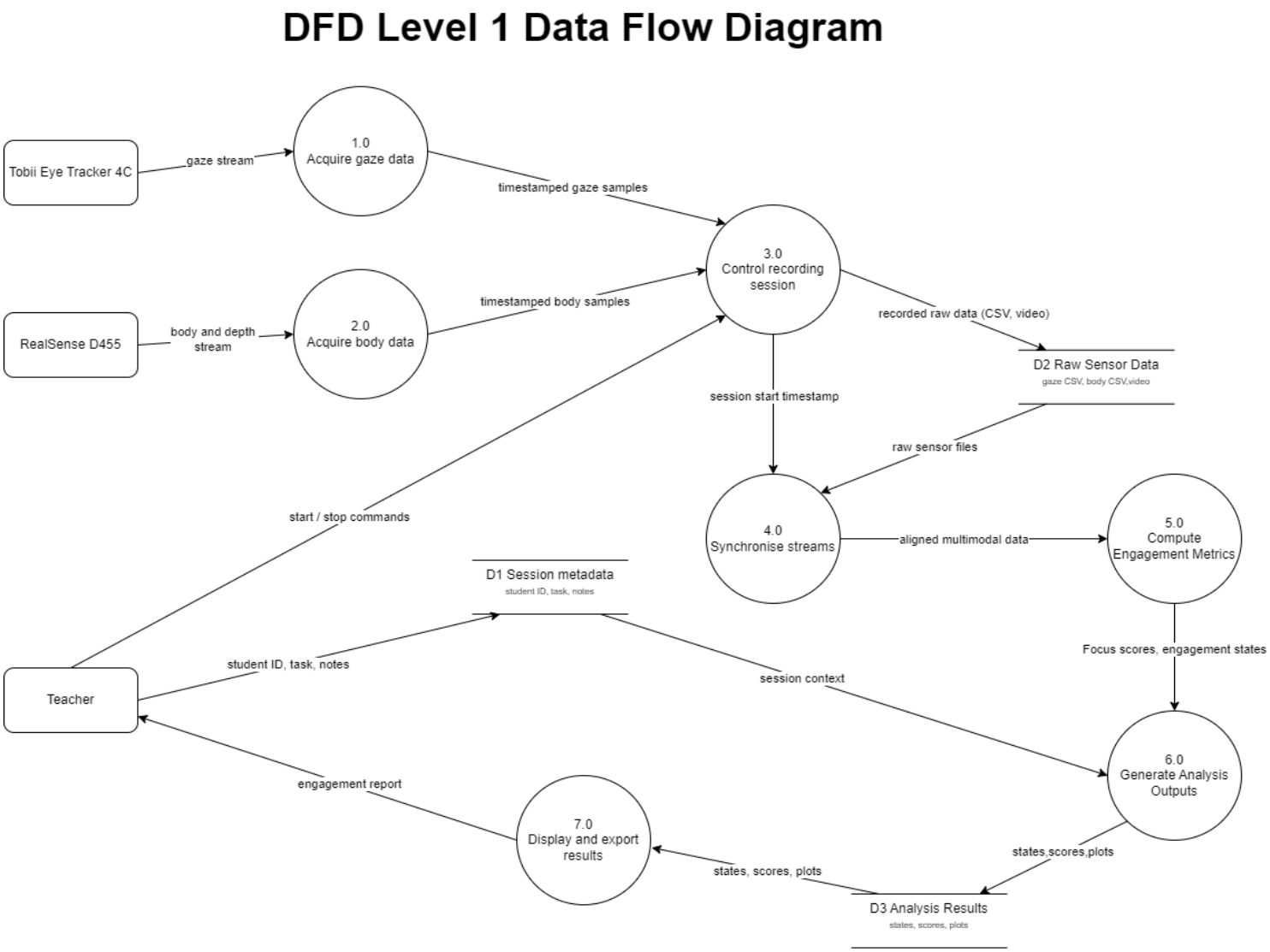


Figure 4– DFD Level 1 showing internal data flow, processes, and data stores of the ALPER-EU system.

3.5. Behavior & Focus Design

This section explains how the system describes and summarizes student engagement during a learning session. The goal is to show how observable signals from gaze and body tracking are turned into clear and understandable engagement indicators, without focusing on technical implementation details.

The behavioral design focuses on clarity, simplicity, and classroom relevance. Engagement results are designed to be easy to understand by teachers and suitable for real educational environments.

3.5.1. Basic Principles of Behavioral Interpretation

The system interprets student behavior using observable signals from gaze and body tracking. Instead of trying to estimate internal thoughts, emotions, or learning outcomes, the system focuses on what can be directly observed during a session (D'Mello & Graesser, 2012; Whitehill et al., 2014).

Three basic principles guide this design:

1. **Observable behavior**: All engagement indicators are based on signals that can be seen and measured, such as where the student is looking or how they move.
2. **Easy interpretation**: Engagement results are designed to be understandable by teachers without technical background.
3. **Classroom context awareness**: The system considers that students may look away from the screen for valid reasons, such as working with a robot.

These principles ensure that engagement results are meaningful and realistic in a classroom setting.

3.5.2. Behavioral Cues from Multiple Sensors

The system uses two main sources of information to describe student behavior: gaze tracking and body tracking. Each source provides different but complementary information.

Gaze data indicate whether the student is looking at the screen or elsewhere and how stable their visual attention is. Body-tracking data describe posture, orientation, and upper-body movement within the workspace.

By combining these two sources, the system can better understand what the student is doing. For example, looking away from the screen while actively moving hands and body toward a robot can indicate task-related activity rather than distraction (Araya & Sossa-Rivera, 2021; Bosch et al., 2015).

3.5.3. Focus Index as an Engagement Summary

The Focus Index is a continuous numerical score, ranging from 0 to 100, that summarizes a student's screen-directed behavioral engagement at any given moment during a session. It quantifies the extent to which a student's observable actions, including gaze direction, body orientation, and physical proximity, are oriented toward the screen-based workstation,

rather than attempting to infer internal cognitive states, motivation, or overall task engagement.

This distinction is intentional. In the educational robotics context, students regularly alternate between screen-based activity and hands-on interaction with a physical robot. During such interaction phases, a student may be fully engaged in the task while their gaze and body are appropriately directed away from the screen. The Focus Index is not designed to capture this form of engagement; instead, it captures the complementary and directly observable dimension of screen-directed workstation attention.

To account for this distinction, disengagement is not inferred from reduced screen orientation alone. Instead, the three-signal disengagement rule described in Section 3.5.4 provides a more conservative mechanism for detecting genuine disengagement, explicitly incorporating the absence of all behavioral signals.

Each signal contributes to the score through a weighted linear combination:

Focus Index = (G × 0.50 + F × 0.30 + D × 0.20) × 100 (Dewan et al., 2019; Pabba & Kumar, 2022)

where **G** represents the gaze signal (1.0 if the student's gaze point falls within the active screen area, 0.0 otherwise), **F** represents the body orientation signal (1.0 if the student is facing forward toward the screen, 0.0 otherwise), and **D** represents the distance signal (1.0 if the student is within the operational range of the depth camera, 0.0 otherwise).

The weights reflect the relative importance of each signal for engagement assessment in the target educational context. Gaze direction is assigned the highest weight (0.50) because it represents the most direct and reliable indicator of visual attention toward the learning task. Research in cognitive science has consistently demonstrated a strong correlation between the direction of eye gaze and the focus of visual attention (Araya & Sossa-Rivera, 2021; Bidwell J. & Fuchs H., 2011). Studies employing automated gaze tracking in authentic classroom settings further confirm that where a student looks is the primary behavioral cue used to infer attentional state, and that attention-based features outperform affective features in engagement classification (Booth et al., 2023; Goldberg et al., 2021; Sümer et al., 2021).

Body orientation is assigned the second-highest weight (0.30) because it captures postural engagement and helps distinguish between on-task and off-task body positioning. Empirical studies indicate that body orientation and head pose exhibit strong consistency with observed engagement states in classroom environments (Araya & Sossa-Rivera, 2021; Chen et al., 2025). In this system, the Facing_Forward signal is determined per frame using shoulder-based orientation cues derived from the relative position and depth of the shoulder landmarks, compared against a calibrated baseline posture for each student.

Physical distance is assigned the lowest weight (0.20) because it functions primarily as a proximity validity signal, confirming that the student remains within the operational range of the Intel RealSense D455 depth camera, rather than serving as a direct measure of cognitive attention (Booth et al., 2023). The combination of multiple behavioral signals through a weighted linear formulation to produce a continuous engagement score is well established in the affective computing and educational technology literature (Booth et al., 2023; Pabba & Kumar, 2022; Sümer et al., 2021).

The score is computed on a per-frame basis at 30 frames per second and subsequently aggregated into one-second intervals, producing a per-second mean score that reduces the influence of momentary signal fluctuations. A centered three-second rolling average is then applied to these per-second values to generate a smoothed engagement score that better reflects sustained engagement trends over time. The use of short temporal windows for aggregating per-frame engagement signals is well established in automated classroom monitoring. Sümer et al. (2021) employ a one-second window for engagement labeling in classroom video analysis, while Alkabbany et al. (2023) demonstrate that short-window aggregation produces stable engagement classifications in STEM classroom environments (Alkabbany et al., 2023; Booth et al., 2023; Sümer et al., 2021). This two-stage smoothing process, consisting of one-second aggregation followed by a three-second centered rolling average, prevents overreaction to brief and expected attention shifts inherent to educational robotics tasks, such as glancing at a physical robot or interacting with input devices, while preserving meaningful engagement trends across the session (Alkabbany et al., 2023; Sümer et al., 2021).

The resulting smoothed score is interpreted across three engagement zones: scores of 80 or above indicate high engagement, scores between 50 and 79 indicate moderate engagement, and scores below 50 indicate low engagement. The use of a three-tier engagement classification, namely high, moderate, and low, is consistent with multiple prior automated engagement systems (Altuwairqi et al., 2021; Reyes et al., 2012; Sümer et al., 2021).

The 80% threshold for the high engagement zone corresponds to students maintaining consistent alignment across all three behavioral signals simultaneously. At a score of 80, the student is directing their gaze toward the screen (G = 1.0, contributing 50 points), facing forward (F = 1.0, contributing 30 points), and remaining within the defined distance threshold (D = 1.0, contributing 20 points).

It should be noted that the Focus Index measures behavioral orientation toward the workstation. During robot interaction phases, when a student is appropriately engaged with physical hardware rather than the display, the score will naturally reflect reduced screen-directed behavior. This constitutes an inherent limitation of gaze-based behavioral proxies in mixed-modality learning environments and is discussed further in the limitations section.

The 50% threshold for the low engagement zone represents the point below which the combined behavioral signals indicate that no dominant engagement cue is consistently present, suggesting the need for teacher attention or instructional adjustment.

These thresholds are designed to support teacher-facing interpretation, presenting engagement levels as clear and categorical descriptions suitable for post-session review, rather than as raw sensor values requiring technical expertise to interpret (Alkabbany et al., 2023; Bidwell J. & Fuchs H., 2011; Goldberg et al., 2021).

The system was preliminarily evaluated in an educational robotics session conducted within the context of the ALPER-EU project at the Hellenic Mediterranean University. During the session, the Focus Index was computed continuously, and the resulting engagement profiles were qualitatively compared with teacher observations and session notes. The session characteristics are comparable to prior automated classroom engagement studies: Sümer et al. (2021) evaluated their system with N = 52 students in 40-minute teaching units, while D'Mello et al. (2014) conducted similar studies with 28 participants in sessions of approximately 30 minutes (D'Mello et al., 2014; Sümer et al., 2021).

The preliminary evaluation indicated that the system produced engagement trajectories consistent with observed classroom behavior, and that the three-signal disengagement rule avoided false disengagement classifications during task-related off-screen activities, such as hands-on interaction with the robot. This behavior is consistent with the conservative design of the rule, where disengagement is only inferred when all three behavioral signals indicate absence of task-related activity (Booth et al., 2023). Qualitative comparison of sensor-derived engagement profiles with direct observation is an established validation approach for classroom monitoring systems (Raca & Dillenbourg, 2013). These findings provide preliminary evidence that the system satisfies *UR-02*, by producing objective and interpretable engagement indicators, and *UR-03*, by correctly handling off-screen behavior in the educational robotics context.

3.5.4. Discrete Behavioral States

In addition to the continuous Focus Index, the system determines a discrete engagement verdict for each recorded frame. This verdict classifies the student as either **Engaged** or **Disengaged** based on a three-signal rule applied to the same behavioral signals used for score computation.

The Three-Signal Disengagement Rule

A student is classified as disengaged only when all three behavioral signals fail simultaneously:

Disengaged = (Gaze OFF-screen) AND (Not Facing Forward) AND (Distance NOT OK)

In all other cases, the student is classified as engaged. This means that the failure of any single signal, or even two signals together, is not sufficient to trigger a disengagement verdict.

This design choice is intentional and reflects the realities of the educational robotics context. A student may look away from the screen while actively typing, glance toward a peer during a collaborative moment, or turn toward a physical robot during a hands-on task. Each of these behaviors, when considered in isolation, is a normal and expected part of task-related activity and should not be interpreted as disengagement.

Disengagement is only inferred when all three behavioral signals are simultaneously absent, namely the absence of visual attention, forward body orientation, and proximity to the workstation, indicating that the student is no longer engaged with the learning task.

Body Engagement as a Combined Postural Signal

A secondary derived signal, Body Engagement, is computed as the conjunction of the orientation and distance signals:

Body Engaged = Facing Forward AND Distance OK

This compound signal captures the postural component of engagement independently of gaze direction. It is used in the analysis outputs to provide a separate view of physical presence and orientation at the workstation, complementing the gaze-based attention signal.

Disengagement Events

Consecutive frames classified as disengaged are grouped into disengagement events. Each event is characterized by its start time, end time, and duration in seconds. Summary statistics, including the total number of events and average event duration, are computed per session and exported alongside the raw engagement data.

This event-based representation allows teachers and researchers to identify not only how much time a student spent disengaged, but also the pattern of disengagement, for example whether it occurred in brief isolated moments or in sustained episodes throughout the session.

Figure 5 illustrates the behavioral logic used for engagement assessment. The system derives two parallel outputs from three behavioral signals (eye gaze, body orientation, and distance). A binary engagement verdict is produced using a conservative rule, where a student is classified as disengaged only if all signals are negative. In parallel, a continuous

engagement score is computed as a weighted combination of the signals and smoothed over time, enabling classification into high, moderate, and low engagement levels.

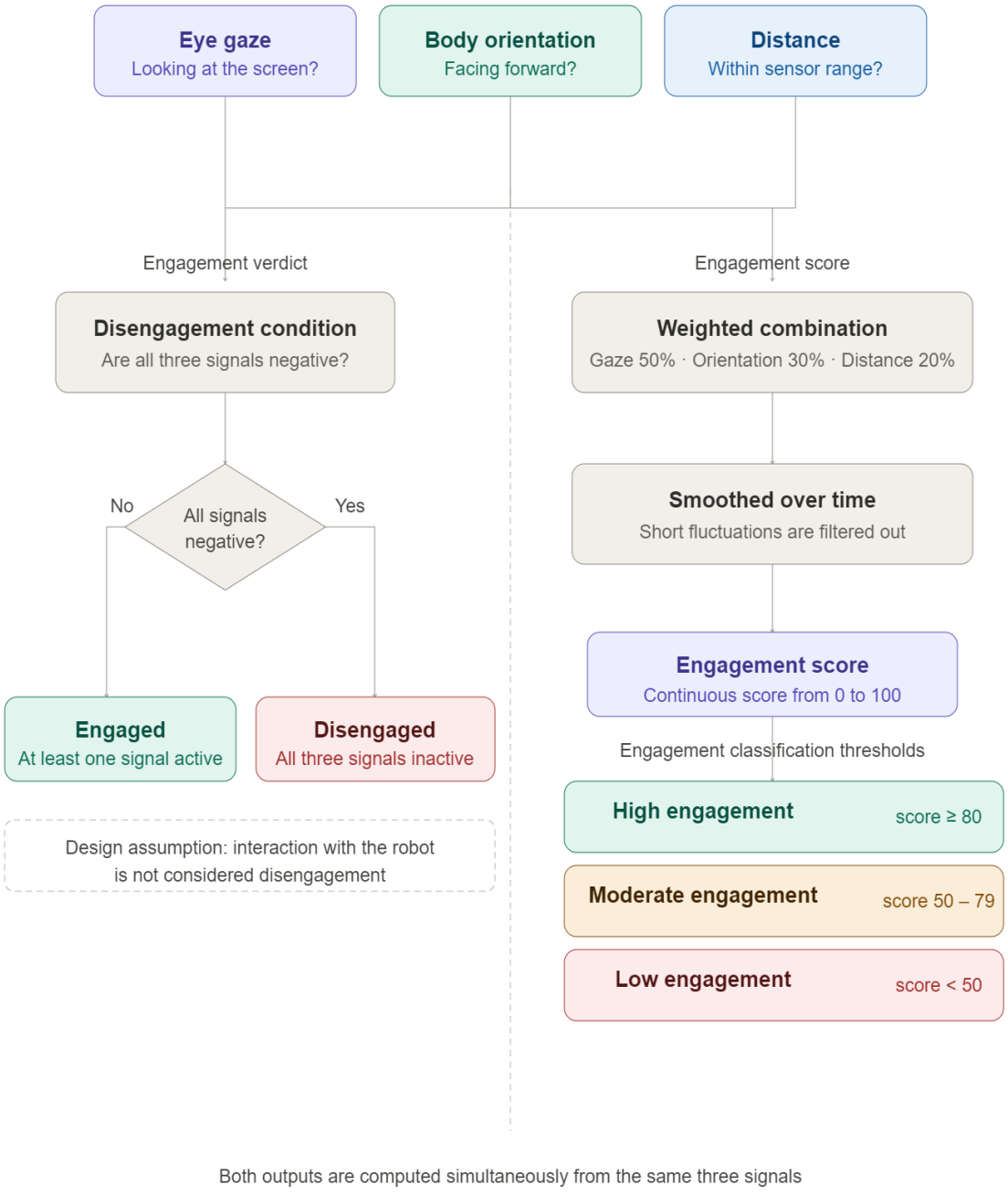


Figure 5 - Behavioral logic for engagement assessment. Three behavioral signals are used to compute both a binary engagement verdict and a continuous engagement score.

3.5.5. Time-Based Interpretation of Behavior

Behavioral states are interpreted over short temporal intervals rather than at individual time instants. This approach reduces the impact of sensor noise, brief postural movements, and momentary attention shifts that do not reflect genuine changes in engagement state (Booth et al., 2023; Sümer et al., 2021). Per-frame classifications are inherently sensitive to such noise, as a student may briefly glance away from the screen, adjust their seated position, or momentarily move outside the camera's optimal range without this constituting meaningful disengagement (Booth et al., 2023; Raca & Dillenbourg, 2013). Evaluating behavior over temporal windows is therefore a well-established practice in automated classroom engagement analysis (Alkabbany et al., 2023; Raca & Dillenbourg, 2013).

In this system, the raw per-frame Engaged or Disengaged verdict is computed at 30 frames per second using the three-signal rule. For the continuous Focus Index, behavioral signals are first aggregated within one-second intervals, followed by the application of a centered three-second rolling average, as described in Section 3.5.3. The use of one-second aggregation aligns with the per-second engagement labeling approach adopted by Sümer et al. (2021), while the three-second smoothing window reflects a minimum duration over which sustained engagement can be reliably distinguished from transient fluctuations (Alkabbany et al., 2023; Sümer et al., 2021).

For discrete disengagement events, consecutive frames classified as disengaged are grouped into contiguous segments, with each event characterized by its start time, end time, and duration in seconds, and exported to `disengagement_events.csv`. The body orientation signal additionally incorporates a frame-level hysteresis mechanism: the Facing_Forward state transitions to False only after 6 consecutive frames satisfy the turn condition, and returns to True after 2 consecutive frames indicate frontal orientation. At 30 frames per second, this corresponds to a minimum persistence of approximately 200 milliseconds before a body turn influences the engagement verdict. This design choice reduces the likelihood of false disengagement classifications caused by brief postural adjustments and is consistent with the use of temporal hysteresis in behavioral state detection for noisy sensor streams (Bidwell J. & Fuchs H., 2011; Raca & Dillenbourg, 2013).

By aggregating observations over time and incorporating temporal hysteresis — specifically a 6-frame entry and 2-frame exit threshold at 30 fps, corresponding to approximately 200 milliseconds and 67 milliseconds respectively — the system produces stable engagement representations that more accurately reflect meaningful classroom behavior and avoid rapid state oscillations that could hinder post-session interpretation by teachers. The use of asymmetric hysteresis, where a longer window is required to enter a new state than to exit it, is a deliberate design choice that reduces false transitions caused by brief postural adjustments while allowing prompt recovery when genuine re-engagement occurs (Booth et al., 2023; Goldberg et al., 2021; Raca & Dillenbourg, 2013).

3.5.6. Teacher-Facing Engagement Representation

A key design goal is to present engagement results in a way that is useful for teachers. Instead of raw sensor values, the system provides clear summaries that describe how engagement changes during a session.

The combination of a continuous Focus Index and discrete behavioral states allows teachers to quickly understand both overall engagement levels and specific types of activity. This supports reflection, discussion, and comparison across sessions without requiring technical expertise (Bidwell J. & Fuchs H., 2011; Goldberg et al., 2021).

3.5.7. Scope and Future Extensions

In the current design phase, behavioral interpretation is based on simple, rule-based logic and observable behavior. More advanced approaches, such as machine-learning models or personalized calibration, are intentionally excluded at this stage (Dewan et al., 2019; Raca & Dillenbourg, 2013).

These features are considered future extensions that can be explored once the current design has been validated and shown to be reliable and useful in classroom environments.

3.5.8. Measurement Scope and Limitations of the Focus Index

The Focus Index is designed as a screen directed behavioral proxy rather than a comprehensive measure of student engagement. This distinction has important implications for how the metric should be interpreted in educational robotics contexts.

First, the metric depends on the type of learning activity at each moment. During screen based phases such as coding, debugging, or reviewing instructions, the Focus Index provides a meaningful summary of workstation oriented attention. During robot interaction phases, however, a student may be fully engaged in the task while their gaze and body are directed away from the screen. In these cases, the Focus Index reflects reduced screen directed behavior, which should not be interpreted as disengagement. Teachers should therefore consider that drops in the score during robot interaction periods may reflect task switching rather than loss of attention.

Second, the Focus Index does not capture cognitive engagement, effort, or learning progress. A student looking at the screen without actively processing content may receive a high score, while a student fully engaged in hands on robot work may receive a lower score. The metric should therefore be interpreted as an indicator of where attention is directed, not how effectively it is used.

Third, the metric has no validated ground truth in this study. The weights assigned to the three signals, 0.50 for gaze, 0.30 for body orientation, and 0.20 for distance, are based on prior literature but have not been calibrated against human annotated data in the ALPER-EU context. Formal validation is therefore identified as future work.

These limitations do not reduce the usefulness of the Focus Index within its intended scope. In Agile Learning sessions, the main activity is screen based coding, while robot interaction phases are shorter and secondary. In this context, the metric provides a consistent, reproducible, and interpretable summary of workstation directed attention that teachers can use without requiring technical expertise.

3.6. Design for Interpretability & Evaluation

This section explains how the system is designed so that its results are easy to understand and can be reliably evaluated. The focus is not on how accuracy is measured in practice, but on how the design itself supports interpretation, verification, and future evaluation.

The goal is to ensure that teachers and researchers can trust the system's outputs and understand how they relate to observable student behavior.

3.6.1. Interpretability as a Core Design Goal

Interpretability is treated as a central design requirement rather than a secondary feature. The system is intentionally designed so that engagement results can be explained using observable signals, such as gaze direction and body posture.

Instead of producing opaque scores or predictions, the system provides outputs that can be directly linked to visible student behavior. This allows teachers to understand *why* a student was considered focused, working on the robot, or distracted, without needing technical knowledge of the underlying system (Booth et al., 2023; Goldberg et al., 2021).

3.6.2. Separation Between Raw Data and Engagement Results

A clear separation is maintained between raw sensor data and derived engagement results. Raw data include gaze coordinates, body joint positions, and synchronized video recordings. Derived results include the Focus Index and discrete behavioral states.

This separation supports transparency and traceability. Teachers and researchers can inspect raw data or video recordings to verify or better understand engagement outcomes.

At the same time, teachers are not required to interact with low-level data during normal classroom use (Alkabbany et al., 2023; Dewan et al., 2019).

3.6.3. Use of Synchronized Video for Validation

Synchronized video recordings play an important role in evaluation readiness. Video provides a direct visual reference that can be compared with inferred engagement states and Focus Index trends.

By aligning video with multimodal sensor data in time, the system enables post-session review and qualitative validation. This design choice allows researchers and educators to check whether engagement interpretations are consistent with what actually occurred during the session.

3.6.4. Session-Based Outputs and Reproducibility

All data and results are organized on a per-session basis. Each recording session produces a self-contained set of outputs, including raw data files, derived engagement metrics, and synchronized video (Raca & Dillenbourg, 2013; Sümer et al., 2021).

This session-based structure supports reproducibility and comparison across different classroom activities. Sessions can be revisited, re-analyzed, or compared over time without ambiguity about data origin or context (Alkabbany et al., 2023; Booth et al., 2023). This design also simplifies future evaluation studies, where consistent data organization is essential.

3.6.5. Support for Future Evaluation and Extension

The current design does not assume a specific evaluation protocol or performance metric. Instead, it provides the necessary foundations for future evaluation by ensuring that data are well-structured, synchronized, and interpretable.

Because engagement inference is rule-based and transparent at this stage, alternative inference strategies or evaluation methods can be applied in later phases without redesigning the entire system. This flexibility allows the system to evolve while preserving comparability with earlier results (Dewan et al., 2019; Raca & Dillenbourg, 2013).

3.6.6. Alignment with Classroom Practice

The design for interpretability and evaluation readiness is closely aligned with classroom practice. Teachers are presented with clear summaries rather than technical metrics, while researchers retain access to detailed data for analysis and validation.

This balance ensures that the system remains usable in real educational settings while still supporting rigorous evaluation and future research.

3.7. Constraints & Ethics by Design

This section describes the practical constraints and ethical considerations that shaped the design of the system. These constraints are not treated as limitations of implementation, but as design choices that ensure responsible use, classroom suitability, and compliance with ethical and legal requirements.

The system is designed for use in real educational environments involving minors. As a result, privacy, transparency, and minimal disruption to classroom activity are central design concerns.

3.7.1. Classroom and Operational Constraints

The system is designed to operate within the practical constraints of a classroom setting. Teachers are not expected to have technical expertise or dedicate time to system configuration during lessons. For this reason, system interaction is limited to simple actions such as starting and stopping a recording session.

The design assumes a single student workstation per session and does not attempt to track multiple students simultaneously. This constraint simplifies system operation and reduces ambiguity in behavioral interpretation. Lighting conditions, camera placement, and workspace layout are assumed to be stable but not tightly controlled, reflecting real classroom conditions rather than laboratory setups.

3.7.2. Non-Intrusive Monitoring by Design

A key design constraint is that monitoring must not interfere with the learning process. The system operates passively once a session has started and does not require user input, wearable devices, or explicit interaction from students.

All sensing is performed using external devices placed in the learning environment. This design minimizes discomfort, reduces distraction, and avoids altering natural student behavior. The system is therefore intended to observe learning activity rather than influence it.

3.7.3. Privacy and Data Protection Considerations

Privacy is a core design concern, particularly because the system may be used with K–12 students. The system is designed to collect only the data necessary for engagement analysis and to avoid unnecessary personal information.

Recorded data are organized per session and can be pseudonymized to prevent direct identification of students. The design supports controlled access to stored data and allows session data to be reviewed, retained, or deleted according to institutional policies.

The system is aligned with general data protection principles, including purpose limitation, data minimization, and transparency. These considerations ensure that the system can be deployed in compliance with data protection regulations such as GDPR (Perrotta & Selwyn, 2020).

3.7.4. Ethical Boundaries of Engagement Interpretation

The system is explicitly designed to avoid inferring internal cognitive, emotional, or psychological states. Engagement is defined in operational terms based on observable behavior, such as gaze direction and body posture.

This design choice reduces the risk of misinterpretation and ethical overreach. Engagement indicators are intended to support reflection and discussion rather than evaluation, grading, or labeling of students. The system does not provide judgments about learning ability, motivation, or performance (Holmes et al., 2022).

3.7.5. Transparency and Teacher Control

Teachers remain in full control of when the system is active and what data are collected. Recording sessions are explicitly started and stopped by the teacher, ensuring awareness and consent at the classroom level.

Engagement outputs are designed to be transparent and explainable. Teachers can relate engagement summaries to observed classroom behavior and, if necessary, review synchronized video recordings for validation. This transparency supports trust and responsible use of the system.

3.7.6. Design Limitations and Responsible Scope

Certain capabilities are intentionally excluded from the current design. The system does not attempt real-time intervention, automated feedback to students, or adaptive behavior based on inferred engagement.

These exclusions reflect a deliberate design boundary aimed at maintaining ethical clarity and avoiding unintended consequences. More advanced features may be explored in future work, following proper evaluation and ethical review.

3.8. Design Phase Summary

This chapter presented the complete design of the ALPER-EU Multimodal Engagement Assessment System, covering the system architecture, synchronization strategy, data flow, behavioral inference model, interpretability considerations, and ethical design constraints.

The system architecture follows a modular and layered structure in which each functional concern, including sensing, synchronization, processing, inference, storage, and user interaction, is assigned to a dedicated layer. This separation ensures that individual components can be developed, tested, and extended independently, which is essential for a research oriented system expected to evolve over time (Booth et al., 2023; Raca & Dillenbourg, 2013).

Temporal synchronization between the two independent sensing pipelines is achieved through a two stage mechanism. A gate based shared recording start ensures that both streams share a common temporal origin, while nearest neighbour alignment with a 50 millisecond tolerance window aligns samples during post session analysis. This design separates real time acquisition from alignment, keeping the recording pipeline lightweight during live operation (Zaletelj & Košir, 2017).

The behavioral inference model is built around a transparent Focus Index that combines three observable signals, gaze direction with weight 0.50, body orientation with weight 0.30, and student proximity with weight 0.20, into a continuous score ranging from 0 to 100. The score is interpreted across three engagement zones: High at or above 80, Moderate between 50 and 79, and Low below 50. A conservative three signal disengagement rule ensures that off screen activity during robot interaction phases is not misclassified as disengagement. The rule based approach was deliberately selected over machine learning methods to ensure transparency, interpretability, and ease of validation at this stage (Dewan et al., 2019; Goldberg et al., 2021).

All design decisions were traced to documented user requirements, identified as UR-01 through UR-07, ensuring that the architecture reflects the pedagogical context and practical constraints of educational robotics classrooms. Ethical considerations, including

data minimization, non intrusive operation, GDPR alignment, and the intentional exclusion of cognitive or emotional state inference, were treated as core design requirements throughout (Holmes et al., 2022; Perrotta & Selwyn, 2020).

The design established in this chapter provides the complete specification from which the implementation presented in Chapter 4 was developed.

4. Implementation Phase of the Multimodal Engagement Assessment System

4.1. Introduction

This chapter presents the practical realization of the ALPER-EU Multimodal Engagement Assessment System. Building directly on the architecture, synchronization strategy, storage design, and behavioral logic defined in the Design Phase, the system is implemented as a modular Python application that integrates gaze data acquisition, body tracking, synchronized multimodal recording, structured session storage, and post-session engagement analysis into a unified and deployable software system.

The implementation follows the layered structure established during the design process. Each major subsystem, including sensor acquisition, temporal synchronization, data storage, engagement inference, and user interaction, is realized as a dedicated module. This approach preserves the separation of concerns that guided the design rationale and enables individual components to be developed, tested, and extended independently. Such modularity is particularly important in the context of a research-oriented system, where iterative refinement and future extensions are expected.

The system is operated through a minimal graphical user interface designed for use in real classroom environments. A recording session is initiated by the teacher through a single action, which triggers the simultaneous activation of all sensing pipelines. From that point onward, data acquisition proceeds automatically until the session is terminated. Upon completion, all raw data files are finalized and stored, and the analysis pipeline can be executed to generate engagement summaries, behavioral timelines, gaze heatmaps, and teacher-oriented reports.

The following sections describe the implementation of each subsystem, beginning with the development environment and software structure, and proceeding through each functional layer in the order in which data flows through the system.

4.2. Development Environment and Tools

This section describes the software and hardware environment used for the implementation of the ALPER-EU Multimodal Engagement Assessment System. The selection of technologies was guided by three primary criteria: compatibility with the sensing hardware, support for real-time multimodal data processing, and ease of integration within a unified development environment. Particular emphasis was placed on minimizing system complexity while ensuring that all functional requirements defined in the Design Phase could be implemented in a reliable and reproducible manner. The implementation environment consists of the Python programming language, a set of specialized libraries for sensor interaction and data processing, and two hardware sensing devices responsible for capturing multimodal behavioral data.

4.2.1. Programming Language and Runtime

The system was implemented in Python (version 3.10 or newer). Python was selected as the primary implementation language due to its extensive support for scientific computing, rapid prototyping, and system integration within a single runtime environment. It is widely adopted in research and engineering applications for handling numerical computation, data analysis, and real-time processing tasks (McKinney, 2010; Oliphant, 2007).

A key advantage of Python in this context is the availability of a comprehensive standard library that directly supports several core system requirements. The graphical user interface is implemented using the tkinter module, which provides a lightweight and portable interface without requiring external dependencies (Python Software Foundation, 2024). Concurrent execution is handled through the threading module, enabling independent sensor pipelines to operate simultaneously within the same process. Thread-safe communication between modules is supported by the queue module, ensuring safe data exchange between concurrent components. High-resolution timestamping is achieved through the time module, specifically using the `time.time_ns()` function, which provides nanosecond precision and enables consistent host-based temporal alignment across all data streams (Python Software Foundation, 2024).

Beyond the standard library, Python's broader ecosystem played a central role in enabling both data processing and visualization functionalities. Libraries such as NumPy and pandas provide efficient structures for handling large time-series datasets, while SciPy and Matplotlib support signal processing and visualization. The ability to implement the entire pipeline—from sensor acquisition to analysis and reporting—within a single language environment significantly reduces system complexity and avoids the need for cross-language integration (McKinney, 2010; Oliphant, 2007). This unified approach is

particularly beneficial in a research-oriented system, where rapid iteration and modular refinement are essential.

4.2.2. Hardware Devices

Two sensing devices were used throughout development and testing, each providing complementary behavioral information.

The Tobii Eye Tracker 4C was used to capture gaze direction and eye position relative to the display. This device enables the estimation of visual attention by determining whether the student is looking at the screen. In the implemented configuration, the system does not interact with the device through a direct Python SDK. Instead, gaze data are published by the TobiiStream local service over a TCP socket, and the application subscribes to this stream at runtime. This architecture allows gaze acquisition to operate independently of the main application loop while maintaining continuous data availability (Tobii AB, 2021).

The Intel RealSense D455 depth camera was used to capture synchronized RGB and depth streams from the student workspace. The device was configured to stream depth frames at 848 × 480 pixels and color frames at 1280 × 720 pixels, both at 30 frames per second. These settings provide a balance between computational efficiency and sufficient spatial resolution for body tracking and video recording. The RealSense SDK enables simultaneous acquisition of both modalities and supports precise timestamping and frame synchronization (Intel Corporation, 2023).

To ensure spatial consistency between the two modalities, depth frames were aligned to the color image coordinate system using the SDK's built-in alignment functionality. This alignment allows depth values to be accurately associated with pixel positions in the color frame, which is essential for computing distance-related behavioral signals based on detected body landmarks. Accurate depth alignment is critical for reliable estimation of posture and proximity, both of which are important indicators of student engagement (Intel Corporation, 2023).

The combined use of gaze tracking and depth-based body tracking enables a multimodal representation of student behavior. This approach has been shown to provide more robust engagement estimation compared to single-modality systems, particularly in classroom environments where behavior is dynamic and context-dependent (Zaletelj & Košir, 2017).

4.2.3. Libraries and Frameworks

The implementation relies on a set of Python libraries that support sensor integration, real-time processing, data storage, analysis, and visualization. All dependencies are installable

via pip, and no additional compilation is required beyond the official SDK bindings for the sensing devices.

The pyrealsense2 library provides Python bindings for the Intel RealSense SDK 2.0 and is used to configure sensor streams, retrieve synchronized frames, and access per-pixel depth values. For body tracking, the system uses MediaPipe, a framework designed for building real-time perception pipelines using machine learning models (Lugaresi et al., 2019). MediaPipe Pose is applied to each color frame to detect body landmarks.

In the production configuration, the pose model is instantiated with `model_complexity=0`, which is the lightest available configuration. This decision is driven by real-time performance constraints. At 30 frames per second, each frame must be processed within approximately 33 milliseconds. Higher-complexity models introduce additional computational overhead that would compromise real-time operation. The selected configuration provides sufficient accuracy for detecting key landmarks, such as the nose and shoulders, which are used for computing orientation and distance signals relevant to engagement estimation (Lugaresi et al., 2019; Zaletelj & Košir, 2017).

Communication with the TobiiStream service is implemented using pyzmq, which provides Python bindings for the ZeroMQ messaging framework. ZeroMQ is designed for high-performance asynchronous communication and supports scalable message-passing patterns (Hintjens, 2013). In this system, a SUB socket is used to receive gaze data in a non-blocking manner. The implementation continuously drains all pending messages before processing each sample, ensuring that the most recent gaze data are always used. This approach prevents the accumulation of outdated values and improves the temporal accuracy of the recorded data.

The opencv-python library is used for image processing, visualization, and video recording. It supports real-time rendering of frames, overlay of detected landmarks, and encoding of video streams using the XVID codec. The use of XVID ensures compatibility across systems and avoids reliance on external encoding tools.

For data handling and analysis, the system uses pandas and numpy. Pandas is used to load CSV files and perform timestamp-based alignment between data streams using functions such as `merge_asof`, while NumPy supports efficient numerical operations. Visualization is performed using matplotlib, which is used to generate engagement timelines, summary charts, and gaze heatmaps (Hunter, 2007). Gaussian smoothing is applied to the gaze heatmap using functions from SciPy to improve interpretability by reducing noise in sparse gaze distributions (Harris et al., 2020; McKinney, 2010).

Additional libraries support the graphical interface and reporting functionality. The Pillow library is used to display generated images within the interface, while openpyxl is used to generate structured Excel reports for teachers. The graphical user interface is implemented

using tkinter, which provides a lightweight and portable solution without introducing external dependencies (Python Software Foundation, 2024).

4.2.4. Development Setup

The system was developed and tested on a Windows 10 workstation. This platform was selected primarily due to compatibility with the Tobii Eye Tracker 4C software stack, which is officially supported on Windows. In addition, the video recording pipeline implemented using OpenCV and the XVID codec is readily supported on Windows systems without requiring additional multimedia frameworks.

The application is executed directly from the command line using a single command:

```
python MainApp.py
```

No compilation or build step is required, and deployment only involves installing the required Python packages and sensor drivers. This simplicity is an important design choice, as it enables the system to be deployed easily in laboratory and classroom environments without requiring advanced technical setup.

Overall, the selected development environment supports the key requirements of the system, including real-time multimodal data acquisition, synchronized processing, structured storage, and post-session analysis. The combination of Python, its scientific ecosystem, and compatible sensing hardware provides a practical and efficient foundation for implementing a research-oriented engagement analysis system.

4.3. Software Structure and Module Organization

4.3.1. Project Directory Layout

The implementation of the ALPER-EU Multimodal Engagement Assessment System is organized as a single project directory containing four primary Python source files together with a small set of supporting analysis utilities. The directory structure reflects the system's functional decomposition into session orchestration, sensor acquisition, and post-session analysis:

```
ALPER-EU/
├── MainApp.py
├── Gaze APP Python/
│   └── GazeAppAlpha.py
├── RealSenseBodyTracker/
│   └── body-tracker.py
├── analysis/
```

```
│   ├── engagement_analysis.py
│   ├── multi_session_comparison.py
│   └── session_exporter.py
└── recordings/
    └── both_YYYYMMDD_HHMMSS/
        ├── tobii_gaze.csv
        ├── upper_body.csv
        ├── hands_data.csv
        ├── realsense_color.avi
        ├── session_metadata.json
        └── analysis/
```

This layout follows the principle of separating concerns across modules, a well-established software engineering practice that reduces coupling, simplifies testing, and makes each component independently understandable (Gamma et al., 1994; Parnas, 1972). Each module is assigned a clearly bounded functional responsibility, and no module depends on the internal implementation details of another. Instead, interaction is limited to explicit public methods, controlled shared state, or file-based outputs, as discussed in Section 3.3.

The directory structure also reflects the distinction between runtime components and post-session processing components. The three runtime modules, namely `MainApp.py`, `GazeAppAlpha.py`, and `body-tracker.py`, participate directly in live session execution. By contrast, the scripts placed in the `analysis/` folder operate on recorded session data after acquisition has completed. This distinction is important from both a software engineering and a research perspective. It ensures that the real-time recording pipeline remains as lightweight as possible during a session, while allowing analysis and reporting logic to evolve independently without affecting raw data collection (Romero & Ventura, 2010; Siemens & Baker, 2012).

Each session produces a self-contained folder under `recordings/`, named using a mode prefix and a timestamp. For example, `both_20260318_143022` denotes a synchronized dual-sensor session started on 18 March 2026 at 14:30:22. The mode prefix, such as `both`, `eye`, or `body`, encodes the sensor configuration used during recording, while the timestamp guarantees uniqueness across sessions. All raw sensor data, metadata, and derived analysis outputs associated with a session are stored within this folder, with analysis products written to a dedicated `analysis/` subfolder.

This per-session directory structure has several important advantages. First, it ensures that sessions are independent and reproducible, since all files required to inspect, reprocess, or validate a given session are stored in one place (Alkabbany et al., 2023; Raca & Dillenbourg, 2013). Second, it preserves a clear distinction between raw data and derived results, which supports data provenance and makes it possible to rerun the analysis pipeline without altering the original recordings (Stodden et al., 2016). Third, it simplifies comparative analysis across sessions, since each session constitutes a clearly defined unit of storage and interpretation.

Overall, the project directory layout is not merely a matter of file organization, but a direct implementation of the system's architectural principles. It supports modularity, reproducibility, and maintainability, all of which are especially important in a research-oriented software system that is expected to be extended and re-evaluated over time.

4.3.2. Role of Each Module

MainApp.py — Application Entry Point and Session Orchestrator

`MainApp.py` serves as the single entry point of the application and is launched through the command `python MainApp.py`. It contains the `MainController` class, which builds the three-tab Tkinter interface and manages the full session lifecycle, from application startup to post-recording verification.

At startup, `MainApp.py` loads the two sensor modules into the same Python process using `importlib.util`. This dynamic loading mechanism is necessary because `body-tracker.py` resides inside the `RealSenseBodyTracker/` subdirectory, which is not structured as a formal Python package and therefore cannot be imported through a standard package import statement. Instead, the file is loaded explicitly by path using `importlib.util.spec_from_file_location()` and `module_from_spec()`, which is the standard Python mechanism for loading modules that are not present on the normal import path (Python Software Foundation, 2024). In practice, this allows the system to preserve its existing directory structure without restructuring the RealSense tracker as an installable package.

Once the modules have been loaded, `MainController` creates one instance of `BodyTrackerEngine` and one instance of `GazeAnalysisApp`, both of which remain active within the same application process. During a recording session, `MainController` is responsible for computing the shared `go_epoch_ms` synchronization gate, distributing this timestamp to both sensing modules before the countdown completes, and coordinating the start and stop of session recording.

A further responsibility of `MainController` is live engagement monitoring. It periodically polls the current engagement-related state from both sensing modules every 200 milliseconds using Tkinter's `root.after()` scheduling mechanism. This polling-based approach ensures that all GUI updates occur on the main thread, which is essential because Tkinter widgets are not thread-safe and must only be manipulated by the thread that owns the event loop (Lundh, 1999; Shipman, 2013). The choice to centralize session orchestration inside `MainController` therefore aligns both with GUI safety requirements and with the broader design principle of maintaining a single coordinating component for runtime control.

GazeAppAlpha.py — Gaze Capture Module and Analysis UI

`GazeAppAlpha.py` implements all Tobii-side functionality and can operate in two distinct modes, depending on how it is instantiated. In embedded mode, when launched by `MainController` with pre-existing Tkinter frames, it builds its widgets directly into the parent application window. In standalone mode, it constructs its own independent Tkinter window with a two-tab notebook. This dual-mode structure reflects the reusable-component pattern, in which the same underlying functionality can be integrated into a larger application or executed independently without code duplication (Gamma et al., 1994).

In both modes, gaze capture is performed by a single daemon thread named `tracking_loop`. Using a daemon thread ensures that the background capture process terminates automatically when the main process exits, thereby preventing the application from hanging during shutdown (Beazley & Jones, 2013). The thread subscribes to the TobiiStream ZeroMQ service, samples gaze data at the configured rate, and begins writing rows to `tobii_gaze.csv` only after the shared `go_epoch_ms` gate has been reached.

The module also supports live visualization. A `queue.Queue` object is used to transfer gaze-related information from the background thread to the GUI thread, following the standard producer-consumer pattern commonly used in Python GUI applications to achieve thread-safe communication (Lutz, 2013; Martelli et al., 2005). In addition, the module maintains a live `gaze_on_screen` Boolean state, which is written by the tracking thread and read by `MainController` during periodic polling. Under CPython's Global Interpreter Lock, reading or writing a single Python object reference is atomic, which makes this usage pattern safe without requiring explicit locking for that specific variable (Beazley, 2010; Python Software Foundation, 2024).

The result is a gaze module that combines real-time capture, CSV logging, and live visualization while remaining sufficiently modular to function either as a standalone application or as an embedded subsystem within the main interface.

body-tracker.py — Body Tracking Engine and Video Recording

`body-tracker.py` implements the RealSense-side functionality through the `BodyTrackerEngine` class, which exposes a clean public API consisting of `start_preview()`, `start_recording()`, `stop_recording()`, and `stop_preview()`. The separation between preview state and recording state is a deliberate implementation decision. Preview begins as soon as the application is ready, before the countdown starts, whereas recording begins only when the shared synchronization gate is reached. This arrangement allows the MediaPipe pose model to warm up and stabilize before any data are written to disk, thereby avoiding the detection instability that can occur in the first frames of a newly initialized inference pipeline (Lugaresi et al., 2019).

The preview loop itself runs in a daemon thread. All per-session state, including the shoulder-width calibration baseline, exponential smoothing dictionaries for landmark coordinates, and the `body engagement signals` dictionary, is stored as instance variables of `BodyTrackerEngine`. This design ensures that each session begins with a clean state and that calibration values from a previous recording cannot silently influence the signals computed in a later session. Such leakage would be especially problematic for body orientation estimation, where stale calibration could produce systematically incorrect engagement judgments.

The `body engagement signals` dictionary is updated by the background thread and read by `MainController` through its polling loop. As with the gaze module, the relevant values are simple Boolean or `None`-type entries, and under CPython's execution model such individual lookups and assignments are safe for this usage pattern without explicit locking (Beazley, 2010). For thread termination, the module uses `threading.Event` rather than a plain Boolean flag. This is an important implementation detail because an event object provides a visibility-safe stop signal across threads and allows the preview loop to react correctly even if it is blocked waiting for a frame from the RealSense pipeline (Python Software Foundation, 2024).

Within the same file, the `SessionLogger` class manages the three output files produced during a session. Because `log_frame()` may be invoked by the preview thread while `close()` may be called by the main thread during session termination, access to the shared `cv2.VideoWriter` object is protected with a `threading.Lock`. This prevents race conditions on a mutable shared resource and follows standard multi-threaded programming practice in Python (Gorelick & Ozsvald, 2020; Martelli et al., 2005).

Taken together, `body-tracker.py` combines frame acquisition, pose inference, behavioral signal generation, and synchronized file logging into a single runtime engine with a well-defined external interface. This makes it possible for the main application to control the RealSense subsystem without depending on its internal implementation details.

engagement_analysis.py — Post-Session Analysis Pipeline

`engagement_analysis.py` implements the post-session analysis stage as a self-contained analysis script that can be run from the terminal or invoked programmatically from the GUI. Its main public interface is the function `run_analysis(folder)`, which loads the session CSV files, merges gaze and body streams using `pandas.merge_asof`, computes the Focus Index and the three-signal engagement verdict, identifies disengagement events, and writes all derived outputs to the session's `analysis/` subfolder. The function returns a `(summary, merged)` tuple, allowing the GUI to display key session statistics directly after analysis without requiring an additional file read.

The decision to keep the analysis pipeline separate from the acquisition pipeline is both architectural and methodological. From a runtime perspective, it ensures that acquisition threads are not burdened by post-processing workload during a live session, thereby protecting the 33 millisecond frame budget required for sustained 30 FPS operation. From a research perspective, it ensures that raw data remain stable and unaffected by future modifications to the inference logic. Analysis can therefore be re-run, refined, or replaced without any risk of altering the original recordings, which is a fundamental requirement for reproducibility (Raca & Dillenbourg, 2013; Stodden et al., 2016).

This separation between raw acquisition and derived analysis is consistent with broader practice in educational data mining and learning analytics, where data collection and interpretation are often treated as distinct pipeline stages in order to preserve auditability and methodological transparency (Romero & Ventura, 2010; Siemens & Baker, 2012). In the present system, this architectural separation also makes the implementation easier to maintain, because changes to engagement logic do not require changes to the runtime recording modules.

session_exporter.py and multi_session_comparison.py — Reporting Utilities

The two remaining scripts in the `analysis/` directory provide supporting reporting functionality.

`session_exporter.py` generates a styled five-sheet Excel workbook using `openpyxl`, intended as a teacher-facing output after a session. The five sheets summarize the session report, per-second engagement scores, raw gaze data, raw body data, and disengagement events with severity classifications. Presenting results in spreadsheet form rather than in lower-level technical formats such as JSON or plain CSV reduces the barrier to use for non-technical stakeholders, particularly teachers, who are more likely to interact comfortably with familiar office-document formats (Goldberg et al., 2021; Prieto et al., 2016).

`multi_session_comparison.py` accepts two or more analyzed session folders and produces side-by-side comparison plots together with a summary table. This supports longitudinal interpretation across sessions, allowing engagement patterns to be compared over time across different students or classroom activities. Such functionality is aligned with the broader goals of multimodal engagement analytics, which often extend beyond single-session inspection toward pattern discovery across repeated educational observations (Booth et al., 2023; Dewan et al., 2019).

Together, these two utilities extend the system beyond raw recording and single-session analysis, enabling teacher-facing reporting and cross-session comparison while remaining cleanly separated from the live acquisition components.

4.3.3. Inter-Module Communication

The three runtime modules, namely `MainApp.py`, `GazeAppAlpha.py`, and `body-tracker.py`, communicate through two primary mechanisms.

The first mechanism is direct method invocation on shared object references. Because all three modules execute within the same Python process, `MainController` holds direct references to both `GazeAnalysisApp` and `BodyTrackerEngine` and calls their public methods directly. This is the simplest and most efficient solution for control signals such as `start_recording()` and `stop_recording()`, since it avoids the serialization overhead and added complexity associated with interprocess communication (Hintjens, 2013; Tanenbaum & Van Steen, 2007). It also makes the runtime control flow explicit in the source code, which is advantageous in a research prototype where clarity, debuggability, and maintainability are especially important (Hunt & Thomas, 1999).

The second mechanism is shared in-process state combined with periodic polling. Rather than using callback-driven synchronization for every live state update, the sensor modules expose a small set of readable live state variables. `GazeAnalysisApp` exposes `gaze_on_screen` as a single Boolean attribute, while `BodyTrackerEngine` exposes `body engagement signals` as a compact dictionary containing three Boolean engagement-related values. `MainController` reads these attributes inside a `_poll_engagement()` method that reschedules itself every 200 milliseconds using `root.after(200, self._poll_engagement)`.

This design is appropriate for the intended purpose of live UI feedback. A 200 millisecond polling interval is sufficiently responsive for engagement display, since visual changes in user interfaces do not need to update at sensor frame rate in order to remain perceptually effective (Card et al., 2018). At the same time, this interval introduces negligible computational overhead relative to the sensor-processing workload. More importantly, polling from the Tkinter main thread ensures that all widget updates occur on the thread that owns the GUI event loop, thereby avoiding thread-safety issues with Tkinter widgets (Shipman, 2013).

The one structural exception to this otherwise unified interface is the body tracker's live OpenCV preview window. Because `cv2.imshow()` cannot be embedded directly inside a Tkinter widget on Windows in a reliable way, the annotated RealSense preview is displayed as a separate operating system window. This is a known practical limitation of the OpenCV and Tkinter combination on Windows and does not affect data acquisition, synchronization, or engagement computation (Bradski & Kaehler, 2008; Intel Corporation, 2023).

Overall, the communication model deliberately prioritizes simplicity over architectural over-engineering. Control signals are handled through direct method calls, lightweight live

state is exposed through readable shared attributes, and all persistent communication between runtime and analysis stages occurs through recorded files in the session directory. This combination is well suited to the needs of a research prototype: it is transparent, sufficiently efficient, easy to debug, and closely aligned with the modular design principles established earlier in the thesis.

Figure 6 illustrates the relationships between the main software modules and their communication mechanisms. At the center of the architecture is the MainController, which orchestrates the session by invoking start and stop operations and receiving status signals from both GazeAnalysisApp and BodyTrackerEngine. Communication between these modules is performed through direct method calls, accompanied by periodic status updates reflecting gaze and body engagement signals.

GazeAnalysisApp receives gaze data externally via a ZeroMQ subscription to the TobiiStream service, while BodyTrackerEngine performs local acquisition and processing of body tracking data. Each module is responsible for recording its own session outputs, with gaze data stored in `tobii_gaze.csv` and body related data stored in `upper_body.csv`, `hands_data.csv`, and recorded video files. Together, these interactions define the software-level control architecture of the implemented system, as illustrated in Figure 6.

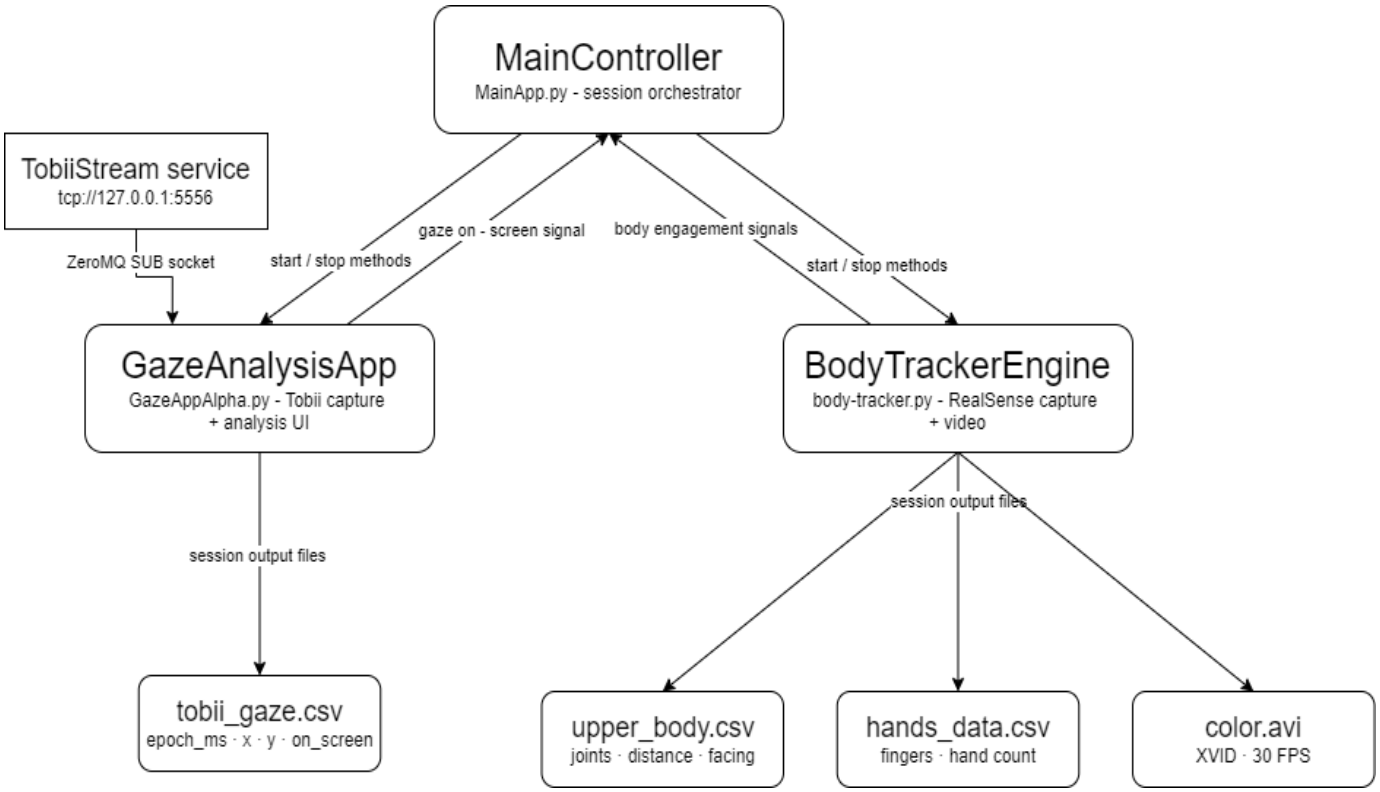


Figure 6 - Module relationships and communication paths in the ALPER-EU system, showing direct method calls, periodic state polling, and per-module session outputs

4.4. Implementation of Gaze Data Acquisition

4.4.1. Connection to the TobiiStream Service

Gaze data acquisition in the implemented system relies on the TobiiStream local service, an external process that interfaces with the Tobii Eye Tracker 4C hardware and publishes raw gaze data over a local TCP socket at tcp://127.0.0.1:5556. This architecture, in which the sensor driver and the application are decoupled through a message-passing interface, follows established patterns for integrating proprietary sensing hardware into research software without binding the application to a specific vendor SDK version (Hintjens, 2013; Tobii AB, 2021).

The application connects to this service using a ZeroMQ SUB socket, which is a subscriber endpoint in the publish-subscribe messaging pattern. On startup, the socket connects to the service address and applies a topic filter set to the string "TobiiStream", ensuring that only gaze-related messages are received and that any future messages published on different topics by the same service are automatically discarded:

```
self.socket = self.context.socket(zmq.SUB)
self.socket.connect("tcp://127.0.0.1:5556")
self.socket.setsockopt_string(zmq.SUBSCRIBE, "TobiiStream")
```

Each message published by TobiiStream is a space-delimited string with the format:

```
TobiiStream <sequence> <raw_x> <raw_y> <validity>
```

where raw_x and raw_y are floating-point gaze coordinates in screen pixels, and validity is an integer flag indicating tracking quality. The application parses these fields by splitting the message string and extracting the relevant values by index.

4.4.2. Two-Loop Sampling Architecture

The core of the gaze acquisition module is a background daemon thread that implements a two-loop design. This architecture separates the concerns of message reception and timed sample production into two independent sub-loops that execute within the same thread iteration (Beazley & Jones, 2013).

The first sub-loop is the **receiver**. At every iteration of the outer loop, it drains all pending ZeroMQ messages in a non-blocking manner using socket.poll(0) followed by socket.recv_string(zmq.NOBLOCK). Draining all available messages before proceeding ensures that the most recently published gaze position is always used for the next sample, regardless of how many messages have accumulated since the previous iteration. This prevents a known failure mode in which the sampler consumes a stale off-screen value

from several frames ago when the student's gaze has already returned to the screen, a problem that would arise from a simple single-receive approach (Hintjens, 2013):

```
while self.socket and self.socket.poll(0):
    try:
        message = self.socket.recv_string(zmq.NOBLOCK)
        parts = message.split()
        if len(parts) >= 4 and parts[0] == "TobiiStream":
            last_tobii_timestamp = time.perf_counter()
            last_raw_x = float(parts[2])
            last_raw_y = float(parts[3])
            last_validity = int(parts[4]) if len(parts) >= 5 else 0
    except (zmq.ZMQError, ValueError):
        break
```

The second sub-loop is the sampler. It fires at fixed intervals determined by the configured frame rate, using time.perf_counter() for precise interval tracking. At each firing, it calls _normalize_gaze_data() on the most recently received raw values and, if recording is active and the synchronization gate has been reached, writes one row to the CSV file. The use of time.perf_counter() for interval management, rather than wall-clock time, is important because it provides a monotonic high-resolution timer that is not subject to the discontinuities that can occur with system clock adjustments (Python Software Foundation, 2024):

```
if now >= next_sample_time:
    sample_interval = 1.0 / float(self.target_fps)
    norm_x, norm_y, on_screen = self._normalize_gaze_data(
        last_raw_x, last_raw_y, last_validity, last_tobii_timestamp)
    sample_epoch_ms = epoch_ms()
    # ... write to CSV if gate reached
    next_sample_time += sample_interval
```

To prevent drift accumulation when the loop falls behind schedule — which can occur under momentary CPU load — the sampler resets next_sample_time to the current time plus one interval whenever the lag exceeds a full sample period. This ensures that a temporary delay does not cause a burst of catch-up samples, which would distort the recorded frame rate.

After each sampler firing, the thread yields by sleeping for at most 5 milliseconds. Without this yield, the tight loop would monopolize the CPU on Windows, where the operating system deprioritizes threads that never yield, resulting in delayed recovery when the gaze returns on-screen (Gorelick & Ozsvald, 2020).

4.4.3. Host Timestamp Assignment

Each gaze sample is timestamped at the moment it is produced by the sampler, not at the moment the raw message was received from TobiiStream. This is a deliberate design decision. The hardware timestamp embedded in the TobiiStream message is not used for the CSV record because it uses a different clock reference than the body tracking pipeline, which would make cross-stream alignment unreliable. Instead, both the gaze module and the body tracking module assign host-based timestamps using the same epoch_ms() function:

```
def epoch_ms() -> int:
    return time.time_ns() // 1_000_000
```

This function returns the number of milliseconds elapsed since the Unix epoch, computed from Python's time.time_ns(). Using nanosecond-resolution integer arithmetic before converting to milliseconds avoids floating-point precision loss and ensures that all timestamps across both streams share a common absolute reference (Python Software Foundation, 2024). The implications of this timestamping strategy for cross-stream alignment are discussed in detail in Section 6.

4.4.4. On-Screen / Off-Screen Classification

Before each sample is written to disk, the raw gaze coordinates are passed through _normalize_gaze_data(), which classifies the gaze as either on-screen or off-screen. The logic is as follows.

First, the method checks whether the TobiiStream service has been silent for more than 500 milliseconds by comparing time.perf_counter() against the timestamp of the last received message. If the silence exceeds this threshold, the gaze is classified as off-screen regardless of the last known coordinates. This threshold was chosen to reliably detect device disconnection or driver failure — conditions under which the last received coordinates would be meaningless — without generating false off-screen classifications during brief natural gaps in the Tobii data stream (Tobii AB, 2021):

```
tracking_lost = (now - last_tobii_timestamp) > 0.5
if tracking_lost:
    return -1, -1, False
```

If the device is active, the classification is based entirely on whether the raw coordinates fall within the screen bounds of 1920 × 1200 pixels. A gaze point is classified as on-screen if and only if $0 \leq x \leq 1920$ and $0 \leq y \leq 1200$. The validity field from the TobiiStream message is not used as the primary gating criterion. This is a deliberate implementation choice, because the Tobii Eye Tracker 4C can transmit validity codes of 1 or 2 — indicating

single-eye or partial tracking — alongside perfectly usable coordinate estimates. Gating strictly on validity would produce spurious off-screen classifications during brief tracking interruptions even when the gaze position is valid (Appel et al., 2018; Tobii AB, 2021):

```
within_bounds = (0 <= raw_x <= 1920) and (0 <= raw_y <= 1200)
return (raw_x, raw_y, True) if within_bounds else (-1, -1, False)
```

4.4.5. Gate-Controlled CSV Recording

The gaze module begins writing samples to `tobii_gaze.csv` only after the shared synchronization gate go_epoch_ms has been reached. This gate is computed by MainController before the session countdown begins and passed to start_recording() as an integer Unix millisecond timestamp. Within tracking_loop, every sample that passes normalization is subject to the following gate check before being written:

```
if (self._record_go_epoch_ms is None
        or sample_epoch_ms >= self._record_go_epoch_ms):
    self._record_writer.writerow(
        [sample_epoch_ms, norm_x, norm_y, on_screen])
```

If go_epoch_ms has not yet been reached, the sample is silently discarded — the tracking thread continues running, keeping the ZeroMQ subscription active and the gaze state up to date, but no data are written. This ensures that the CSV file begins at the same absolute moment as the body tracking CSV, which is essential for the nearest-neighbour alignment performed during post-session analysis (McKinney, 2010). The synchronization mechanism itself is described in full in Section 4.6.

To protect against data loss in the event of an unexpected application crash, the CSV file is flushed to disk every 30 rows:

```
self._record_flush_counter += 1
if self._record_flush_counter >= 30:
    self._record_file.flush()
    self._record_flush_counter = 0
```

At 30 FPS, this results in a disk flush approximately once per second, which limits the maximum data loss from a crash to roughly one second of recording while avoiding the performance cost of flushing on every write (Gorelick & Ozsvald, 2020).

4.4.6. Output Format

Each row written to `tobii_gaze.csv` contains four fields, as shown in Table 7.

Table 7 - Schema of the tobii_gaze.csv file containing gaze coordinates, timestamps, and on-screen validity flags.

Column	Type	Description
epoch_ms	integer	Host timestamp in milliseconds since Unix epoch
x	float	Gaze x-coordinate in screen pixels (−1 if off-screen)
y	float	Gaze y-coordinate in screen pixels (−1 if off-screen)
on_screen	boolean	True if gaze is within screen bounds and device is active

Off-screen samples are recorded with coordinate values of −1 rather than being omitted. This preserves a complete and uniform time series at the configured frame rate, which simplifies downstream temporal alignment and ensures that the row count of `tobii_gaze.csv` accurately reflects the session duration (McKinney, 2010).

4.4.7. Configurable Frame Rate

In synchronized dual-sensor recording mode, the gaze module is locked to 30 FPS by MainController, which passes `target_fps=30` to `start_recording()`. This matches the frame rate of the RealSense body tracking pipeline and is the rate used for all analysis. In standalone eye-only mode, the frame rate is selectable via a dropdown in the Live Tracking tab, supporting values from 30 up to 90 FPS. The Tobii Eye Tracker 4C natively publishes at up to 90 Hz, so higher rates are available when body tracking synchronization is not required (Tobii AB, 2021). The `target_fps` value is read dynamically by `tracking_loop` at each sampler firing, meaning that changes to the selector take effect immediately without requiring the thread to be restarted.

4.5. Implementation of Body Tracking and Video Recording

4.5.1. RealSense Pipeline Initialization

The body tracking pipeline is implemented within the `_preview_loop()` method of `BodyTrackerEngine`, which runs in a background thread started by `start_preview()`. At the beginning of each preview session, the method initializes the RealSense SDK pipeline and configures two synchronized streams:

```
pipeline = rs.pipeline()
config = rs.config()
config.enable_stream(rs.stream.depth, 848, 480, rs.format.z16, 30)
config.enable_stream(rs.stream.color, 1280, 720, rs.format.bgr8, 30)
pipeline.start(config)

align = rs.align(rs.stream.color)
```

The depth stream is configured at 848 × 480 pixels in 16-bit unsigned integer format (z16), which provides millimetre precision distance values per pixel. The color stream is configured at 1280 × 720 pixels in BGR format at 30 frames per second. These resolutions were chosen to balance computational efficiency with sufficient spatial detail for both pose estimation and video recording (Intel Corporation, 2023).

After starting the pipeline, depth frames are spatially aligned to the color frame coordinate system using `rs.align(rs.stream.color)`. This alignment step is essential because the depth and color sensors on the D455 have physically separate optical centers, meaning that without alignment, a pixel coordinate in the color image would correspond to a different physical point than the same coordinate in the depth image. Alignment ensures that depth values can be accurately queried at the normalized landmark coordinates produced by MediaPipe, which are computed relative to the color frame (Intel Corporation, 2023; Zaletelj & Košir, 2017).

4.5.2. Pose and Hand Landmark Detection

At each frame iteration, the preview loop retrieves a synchronized aligned frame pair and converts the color image from BGR to RGB, as required by MediaPipe:

```
frames = pipeline.wait_for_frames()
aligned = align.process(frames)
color_frame = aligned.get_color_frame()
depth_frame = aligned.get_depth_frame()

color_image = np.asanyarray(color_frame.get_data())
image_rgb = cv2.cvtColor(color_image, cv2.COLOR_BGR2RGB)
image_rgb.flags.writeable = False

pose_results = pose.process(image_rgb)
```

Setting `flags.writeable = False` before passing the image to MediaPipe avoids an internal copy inside the framework, providing a small performance improvement at 30 FPS where each millisecond contributes to meeting the 33 ms frame budget (Lugaresi et al., 2019).

The pose model is configured with `model_complexity=0`, `min_detection_confidence=0.5`, and `min_tracking_confidence=0.5`. This is the lightest available MediaPipe Pose configuration and was selected specifically to stay within the per frame processing budget.

The shoulder and nose landmarks produced by this model are sufficient for all engagement signals required by the system, making higher complexity models unnecessary (Lugaresi et al., 2019; Zaletelj & Košir, 2017).

Hand inference is handled separately using MediaPipe Hands, configured with `model_complexity=0`, `min_detection_confidence=0.3`, `min_tracking_confidence=0.3`, and `max_num_hands=2`. Hand inference is more computationally expensive than pose inference and is throttled to run every three frames, a constant defined as `HANDS_EVERY = 3`. At 30 FPS this produces hand updates approximately ten times per second, which is sufficient for detecting the presence or absence of hand activity relevant to the engagement model. Between throttled frames, the result from the most recent hand inference is reused:

```
HANDS_EVERY = 3

if frame_counter % HANDS_EVERY == 0:
    hands_results = hands_model.process(image_rgb)

frame_counter += 1
```

This throttling strategy preserves real time frame rate without sacrificing the behavioral information needed for engagement analysis (Gorelick & Ozsvald, 2020).

4.5.3. Landmark Extraction and Depth Estimation

When the pose model successfully detects landmarks, the system extracts three key points: the nose (landmark index 0), the left shoulder (index 11), and the right shoulder (index 12). These are the only landmarks required to compute the distance and orientation signals used in the engagement model, consistent with the design rationale (Asteriadis et al., 2013; Zaletelj & Košir, 2017).

Each landmark is provided by MediaPipe as normalized coordinates in the range [0, 1] relative to the color frame dimensions. To obtain the corresponding depth value, the `get_depth_distance()` function converts normalized coordinates to pixel coordinates and queries the aligned depth frame:

```
def get_depth_distance(x_pixel, y_pixel, depth_frame):
    width = depth_frame.get_width()
    height = depth_frame.get_height()

    x = int(x_pixel * width)
    y = int(y_pixel * height)

    if 0 <= x < width and 0 <= y < height:
        return depth_frame.get_distance(x, y)

    return 0
```

The function returns the depth in metres at the queried pixel. A return value of 0 indicates either an out of bounds coordinate or an invalid depth measurement, both of which are handled gracefully in the downstream signal computation. Depth queries directly on the RealSense SDK are computationally inexpensive and introduce negligible latency to the frame loop (Intel Corporation, 2023).

Exponential smoothing is applied to all three landmark positions and their corresponding depth values using an alpha of 0.4. This reduces the effect of frame to frame jitter in the MediaPipe output, which can produce noisy engagement signal transitions even when the student's actual posture is stable:

```
def smooth(prev, new, alpha=0.4):
    if prev is None:
        return new
    return prev * (1 - alpha) + new * alpha
```

4.5.4. Distance Estimation

The distance signal (`Distance_OK`) indicates whether the student is within a usable working range of the sensor. It is computed from the depth value at the nose landmark, which provides a reliable estimate of the student's distance from the camera. The default distance threshold is 0.70 metres, adjustable at runtime via a slider in the OpenCV preview window. A student is classified as within range when the nose depth reading falls within the configured threshold (Zaletelj & Košir, 2017).

4.5.5. Body Orientation Detection

The facing forward signal (`Facing_Forward`) determines whether the student is oriented toward the screen or has turned away. Rather than relying on a single measurement, the system uses a three cue rotation score that combines complementary signals from the shoulder geometry. This approach is more robust to noise than any individual cue alone and reduces sensitivity to transient fluctuations (Asteriadis et al., 2013; Zaletelj & Košir, 2017).

The three cues are evaluated as follows. First, the shoulder width ratio compares the current horizontal distance between the left and right shoulder landmarks against a calibrated baseline width. If the current width falls below 75 percent of the baseline, it indicates that the shoulders have rotated away from the camera, contributing a score of +1. Second, the depth asymmetry computes the absolute difference in depth between the left and right shoulder measurements. An asymmetry greater than 10 centimetres suggests that one shoulder is significantly further from the camera than the other, again indicating rotation, contributing +1. Third, the height asymmetry evaluates the vertical position difference

between the two shoulders. A difference exceeding 3 percent of the frame height indicates a postural tilt consistent with turning, contributing +1.

If the cumulative rotation score reaches the default turn threshold of 2, the student is classified as turned away. To prevent rapid flickering at the boundary, the classification uses asymmetric hysteresis implemented through a turn counter. The counter increments when a turn is detected and decrements by 2 when the student faces forward again, requiring a sustained return to frontal posture before the state transitions back:

```
if body_turned:
    shoulders.turn_counter = min(MAX_TURN_COUNTER, shoulders.turn_counter + 1)
else:
    shoulders.turn_counter = max(0, shoulders.turn_counter - 2)

if shoulders.turn_counter >= TURN_ENTER_FRAMES:
    facing_forward = False
elif shoulders.turn_counter <= TURN_EXIT_FRAMES:
    facing_forward = True
```

The `Body_Engaged` signal is then defined as the logical conjunction of `Distance_OK` and `Facing_Forward`, and all three signals are passed to `SessionLogger` for recording and to `live_state` for real time display in the GUI.

4.5.6. Dual Timestamping

Each frame is timestamped with two values before being passed to `SessionLogger`:

```
primary_epoch_ms = epoch_ms()
device_ms_val = frames.get_timestamp()
```

The primary timestamp is computed using the shared `epoch_ms()` function at the moment the frame is retrieved. This host based timestamp is written to `upper_body.csv` and used for cross stream alignment. The secondary timestamp is retrieved from the RealSense SDK using `frames.get_timestamp()` and represents the hardware clock of the sensor at the time of frame capture.

The use of the host based timestamp as the primary synchronization reference, rather than the hardware device timestamp, ensures that both streams share the same absolute time reference, which is the prerequisite for the nearest neighbour alignment performed during analysis (McKinney, 2010).

4.5.7. Gate Controlled CSV Recording

Body tracking data are written to the CSV files only after the shared `go_epoch_ms` gate has been reached, mirroring the gate mechanism used in the gaze module:

```
if (self._is_recording
        and self.go_epoch_ms is not None
        and primary_epoch_ms >= self.go_epoch_ms):

    cv2.circle(image_bgr, (30, 30), 10, (0, 0, 255), -1)
    cv2.putText(image_bgr, "REC", (50, 40), ...)

    self.logger.log_frame(
        image_bgr, primary_epoch_ms, device_ms_val,
        pose_results.pose_landmarks if pose_results else None,
        hand_landmarks, depth_frame,
        distance_ok=distance_ok,
        facing_forward=facing_forward,
        body_engaged=body_engaged,
    )
```

The red circle and "REC" annotation are added to the frame before it is passed to `log_frame()`, ensuring that every recorded video frame carries a visible recording indicator. Each call to `log_frame()` writes one row to `upper_body.csv`, one row to `hands_data.csv`, and one frame to `realsense_color.avi`. All file handles are flushed after every write, trading a small I/O overhead for protection against data loss from unexpected termination (Gorelick & Ozsvald, 2020).

4.5.8. Video Recording

```
fourcc = cv2.VideoWriter_fourcc(*"XVID")
self.out = cv2.VideoWriter(self.video_path, fourcc, 30.0, (1280, 720))
```

The XVID codec was selected because it is natively supported by OpenCV on Windows without requiring FFmpeg compilation, ensuring reliable deployment across laboratory workstations. The XVID and AVI combination provides manageable file sizes while maintaining broad playback compatibility (Bradski & Kaehler, 2008; Intel Corporation, 2023).

Access to the `VideoWriter` object is protected by a `threading.Lock` to prevent race conditions between the preview thread and the main thread during session termination.

4.5.9. Output Files

Each active recording session produces three output files from the body tracking module: `upper_body.csv`, which contains per frame pose landmarks and derived behavioral signals, `hands_data.csv`, which contains per frame hand detection results, and `realsense_color.avi`, which contains the annotated synchronized video recording.

Table 8 describes the schema of `upper_body.csv`, which records pose landmarks, derived behavioral signals, and dual timestamps for each frame in which MediaPipe successfully detects a pose.

Table 8 - Schema of upper_body.csv

Column	Type	Description
epoch_ms	integer	Host timestamp in milliseconds since Unix epoch
device_ms	float	RealSense hardware timestamp in milliseconds
Nose_X, Nose_Y	float	Normalized nose landmark coordinates [0, 1]
Nose_Dist_M	float	Nose depth in metres
L_Shoulder_X, L_Shoulder_Y	float	Normalized left shoulder coordinates
L_Shoulder_Dist_M	float	Left shoulder depth in metres
R_Shoulder_X, R_Shoulder_Y	float	Normalized right shoulder coordinates
R_Shoulder_Dist_M	float	Right shoulder depth in metres
Distance_OK	boolean	True if student is within the distance threshold
Facing_Forward	boolean	True if student is oriented toward the screen
Body_Engaged	boolean	True if both conditions are satisfied

When no pose landmarks are detected, for example when the student moves outside the camera field of view or occludes their upper body, no row is written to `upper_body.csv`, resulting in a natural gap in the time series. These gaps are handled during post session alignment using nearest neighbour matching with `pandas.merge_asof`.

Table 9 describes the schema of `hands_data.csv`, which records hand detection results at the same frame rate. Hand inference is executed every third frame, as described in Section 4.5.2, so this file may contain fewer rows than `upper_body.csv` over the same recording interval.

Table 9 - Schema of hands_data.csv

Column	Type	Description
epoch_ms	integer	Host timestamp in milliseconds since Unix epoch
device_ms	float	RealSense hardware timestamp in milliseconds
Fingers_Detected	boolean	True if at least one hand landmark set is detected
Hand_Count	integer	Number of hands detected (0, 1, or 2)

The third output is the synchronized video file `realsense_color.avi`. This file is written using a `cv2.VideoWriter` configured with the XVID codec at 1280 × 720 resolution and 30 frames per second. Each frame is annotated with pose landmarks, the current Distance_OK and Facing_Forward status, and a red REC indicator that appears only after the synchronization gate is reached and recording begins.

The video provides a continuous visual reference for post session validation, allowing engagement outputs to be compared directly with observed classroom behavior (Raca & Dillenbourg, 2013; Stodden et al., 2016).

4.6. Implementation of Synchronization

4.6.1. The Synchronization Problem

The ALPER-EU system integrates two independent sensing pipelines that operate on different hardware platforms, at different sampling rates, and with different initialization latencies. The Tobii Eye Tracker 4C provides gaze data through an external service at up to 90 Hz, while the Intel RealSense D455 body tracking pipeline operates at 30 frames per second, driven by the camera hardware clock. Since these two sensors operate independently and do not share a common timing mechanism, it is necessary to establish an explicit synchronization strategy to ensure that their outputs correspond to the same temporal window and can be reliably associated during analysis.

The implemented solution follows a two-stage approach. The first stage ensures that both pipelines begin recording at a shared temporal reference. The second stage performs alignment of individual samples during post-session analysis using nearest-neighbour matching. These stages are independent and are described in the following subsections.

4.6.2. Unified Host-Based Timestamping

The synchronization strategy is based on a unified timestamping convention applied consistently across both sensing modules. Instead of relying on hardware timestamps from each device, both modules assign timestamps using the same host-side function:

```
def epoch_ms() -> int:
    return time.time_ns() // 1_000_000
```

This function returns the number of milliseconds elapsed since the Unix epoch, using nanosecond-resolution system time. The use of integer arithmetic prior to conversion avoids floating-point rounding errors and ensures consistent timestamp precision across modules (Python Software Foundation, 2024).

Both `GazeAppAlpha.py` and `body-tracker.py` use this identical function. As a result, all entries in `tobii_gaze.csv` and `upper_body.csv` share a common absolute time reference. This consistency is essential for cross-stream alignment.

The RealSense device timestamp, obtained through `frames.get_timestamp()` and stored in the `device_ms` column, is preserved for reference but is not used for synchronization. Device timestamps are based on independent clock domains and cannot be directly compared. Using host-assigned timestamps eliminates discrepancies caused by clock drift or unsynchronized time bases (Tanenbaum & Van Steen, 2007).

4.6.3. Gate-Based Synchronized Recording Start

The first stage of synchronization ensures that both sensing pipelines begin writing data at the same absolute moment. This is achieved through a shared gate timestamp, `go_epoch_ms`, computed by the main application before recording begins:

```
go_epoch_ms = epoch_ms() + secs * 1000
```

The variable `secs` represents the countdown duration selected by the user. The resulting value is a future Unix timestamp that defines the exact moment at which recording should begin.

This same value is passed to both sensing modules:

```
self.gaze_app.start_recording(gaze_csv, go_epoch_ms, target_fps=30)
self.body_engine.start_recording(folder, go_epoch_ms)
```

Each module independently evaluates the current host time during its acquisition loop and begins writing data only when the timestamp condition is satisfied:

- In `GazeAppAlpha.py`, recording begins when `sample_epoch_ms >= self._record_go_epoch_ms`
- In `body-tracker.py`, recording begins when `primary_epoch_ms >= self.go_epoch_ms`

Since both modules rely on the same clock and the same threshold value, their recordings start at the same absolute time. This approach avoids the need for hardware synchronization signals and ensures consistent alignment regardless of differences in initialization time.

During the countdown period, both pipelines are already active. The RealSense pipeline is capturing frames and running pose estimation, while the Tobii subscription is receiving gaze messages. Only file writing is delayed. This ensures that both pipelines are fully initialized and stable when recording begins, which avoids transient inaccuracies during early frames (Lugaresi et al., 2019).

4.6.4. Nearest-Neighbour Temporal Alignment

Because the two sensors operate at different sampling rates and do not produce samples at identical timestamps, direct alignment based on exact timestamp matching is not feasible. Instead, alignment is performed during post-session analysis using nearest-neighbour matching with `pandas.merge_asof` (McKinney, 2010):

```
MERGE_TOLERANCE_MS = 50

merged = pd.merge_asof(
    body,
    gaze[['epoch_ms', 'on_screen', 'x', 'y']],
    on='epoch_ms',
    direction='nearest',
    tolerance=MERGE_TOLERANCE_MS,
)
```

The body tracking stream is used as the reference dataset. For each body frame, the gaze sample with the closest timestamp is selected. The use of `direction='nearest'` allows matching with both earlier and later gaze samples, improving alignment accuracy compared to unidirectional matching (McKinney, 2010; Zaletelj & Košir, 2017).

The tolerance parameter is set to 50 milliseconds. At 30 frames per second, each frame corresponds to approximately 33 milliseconds. A 50 millisecond window therefore spans approximately one and a half frames. This tolerance accommodates normal scheduling delays and minor timing variations while preventing incorrect associations between samples that are temporally distant (Zaletelj & Košir, 2017).

If no gaze sample is found within the specified tolerance window, the corresponding values are assigned conservative defaults:

```
merged['on_screen'] = merged['on_screen'].fillna(False)
merged['x'] = merged['x'].fillna(-1)
merged['y'] = merged['y'].fillna(-1)
```

Assigning `on_screen = False` ensures that missing gaze data is not interpreted as valid attention. This conservative treatment prevents inflation of engagement measures and is consistent with standard practice in multimodal behavioral analysis (Booth et al., 2023; D'Mello et al., 2017).

4.6.5. Adequacy of the Synchronization Approach

The synchronization strategy used in this system is designed to be practical and sufficient for the requirements of classroom-based engagement analysis. It does not rely on hardware triggers, pulse-based synchronization, or network time protocols. Instead, it leverages the fact that both sensing pipelines operate on the same host machine and therefore share a common system clock.

The gate-based mechanism ensures that both pipelines begin recording at a precisely defined time, independent of thread scheduling. The use of host-based timestamps ensures consistency across all recorded data. The nearest-neighbour alignment stage compensates for residual timing differences that arise during execution.

At 30 frames per second, the temporal resolution of the system is approximately 33 milliseconds. The synchronization gate aligns recording start times with sub-millisecond precision, while the alignment stage tolerates differences of up to 50 milliseconds. This range is sufficient to absorb typical operating system scheduling delays, which are generally between 1 and 15 milliseconds on standard desktop systems (Python Software Foundation, 2024; Tanenbaum & Van Steen, 2007).

An additional advantage of this approach is the separation between acquisition and analysis. During recording, both pipelines operate independently without synchronization overhead, which preserves real-time performance. Alignment is performed later during analysis, allowing the process to be applied consistently and reproducibly without affecting data collection. This separation supports reproducibility and aligns with established practices in experimental system design (Raca & Dillenbourg, 2013; Stodden et al., 2016).

4.7. Implementation of the Graphical User Interface

4.7.1. Window Structure and Design Rationale

The graphical user interface is implemented in `MainApp.py` using Python's tkinter library and its themed widget extension `ttk`. The application presents a single unified window organized into three tabs via a `ttk.Notebook` widget. The three tabs are Session Control, Live Tracking, and Data Analysis. This tabbed structure keeps all functionality within one window without cluttering the teacher-facing interface with sensor-specific controls that are not needed during normal classroom operation (Goldberg et al., 2021; Prieto et al., 2016).

The window is initialized at 1050 × 820 pixels with a minimum size of 900 × 700 pixels to ensure all controls remain accessible without scrolling on standard laboratory displays. The title bar reads "ALPER-EU Multimodal Engagement Recorder" to make the application immediately identifiable in a multi-application classroom environment.

Tkinter was selected as the GUI framework because it is included in the Python standard library, requires no additional installation, and runs natively on Windows without external dependencies. Although it is less visually modern than frameworks such as PyQt or wxPython, it is sufficient for the functional requirements of the system, including session control, live status display, and post-session analysis, and avoids adding external dependencies to the deployment (Lundh, 1999; Shipman, 2013).

4.7.2. Session Control Tab

The Session Control tab is the primary interface used by teachers during a recording session. It is built entirely within `MainController._build_session_tab()` and contains four functional areas arranged vertically: countdown configuration, recording controls, sensor status and live engagement monitor, and post-recording verification.

Countdown configuration. A `ttk.Spinbox` widget allows the teacher to set a countdown duration between 0 and 60 seconds. When a session is started, both sensor pipelines enter preview mode immediately while the countdown runs. Recording begins only when the countdown expires and the `go_epoch_ms` gate is reached. A label next to the spinbox explains this behavior in plain language, reducing the need for a manual or training session (Bidwell J. & Fuchs H., 2011).

Recording controls. Three start buttons are provided: Start BOTH, Start Eye Only, and Start Body Only. Each button is mutually exclusive. Once a recording mode is active, the other buttons are disabled. The Start BOTH button is the primary control for synchronized dual-

sensor recording. Before any recording begins, a modal `SessionMetadataDialog` window appears, prompting the teacher to enter a student identifier, task name, facilitator name, and optional notes. A "Guest / Skip" button bypasses this dialog for demos and testing. This metadata is written to `session_metadata.json` in the session folder immediately after the dialog is confirmed:

```
def _on_guest(self):
    self.metadata = {
        'student_id':  'GUEST',
        'task_name':   'Demo / Test',
        'recorded_at': datetime.now().isoformat(),
    }
    self.submitted = True
    self.destroy()
```

A corresponding Stop button for each mode stops both sensor pipelines, finalizes all output files, and triggers the post-recording verification panel automatically after a 500 ms delay to allow file handles to close cleanly.

Countdown display. A large label in the center of the controls area shows the countdown value in red, transitioning to a green "▶ Recording" label when the gate is reached. This provides immediate visual feedback that recording has started, which is important in a classroom context where the teacher may be moving around the room (Goldberg et al., 2021).

4.7.3. Session Metadata Collection

The `SessionMetadataDialog` class is a modal `tk.Toplevel` window that blocks interaction with the parent window until the teacher confirms or cancels. It collects four fields: Student ID (required), Task Name (required), Facilitator (optional), and Session Notes (optional). If either required field is left empty and the teacher presses Start Recording, an inline error label highlights the missing field without dismissing the dialog. The teacher can correct the entry without restarting the process:

```
if not student_id:
    self._error_label.config(text="⚠ Student ID is required.")
    self._entries['student_id'].focus_set()
    return
```

The dialog is centered over the parent window using Tkinter geometry calculations and uses `grab_set()` and `wait_window()` to enforce modal behavior. After the dialog closes, the calling code checks `dialog.submitted` before proceeding, ensuring that no session folder is created and no sensors are started if the teacher pressed Cancel (Shipman, 2013).

4.7.4. Live Engagement Monitor

During an active recording session, the Session Control tab displays a live engagement monitor panel that updates every 200 milliseconds via `root.after()`. The panel shows four indicator labels corresponding to the three behavioral signals and the overall session timer:

Table 10 - Live engagement monitor indicators, associated signal sources, and their corresponding system states.

Indicator	Signal source	True state	False state
Gaze on screen	`gaze_app.gaze_on_screen`	ON-SCREEN ✓	OFF-SCREEN ✗
Within distance	`body_engine.live_state["distance_ok"]`	OK ✓	TOO FAR ✗
Facing forward	`body_engine.live_state["facing_forward"]`	FORWARD ✓	TURNED ✗
Session time	`time.time() - _session_start_time`	MM:SS elapsed	—

A fifth indicator, the overall engagement verdict, is derived from the three signal values according to the conservative disengagement rule. A student is classified as disengaged only when all three signals are simultaneously false. In all other cases, the verdict shows "✓ ENGAGED" in green. When all three fail simultaneously, it shows "⚠ DISENGAGED" in red:

```
all_false = (gaze_on is False
             and dist_ok is False
             and facing_fw is False)

if all_false:
    self.ind_engaged.config(text="⚠ DISENGAGED", background="#c0392b")
else:
    self.ind_engaged.config(text="✓ ENGAGED", background="#27ae60")
```

When any signal value is `None`, which occurs before the sensor pipeline has produced its first frame, the verdict displays "WAITING…" in gray, preventing misleading engagement classifications during initialization (Card et al., 2018).

4.7.5. Post-Recording Verification Panel

After a session stops, the `verify_recording()` method runs automatically and populates a verification panel in the Session Control tab. This panel reads the three CSV files and the video file from the session folder and reports the row count of each CSV, the estimated frame rate computed from the median inter-timestamp difference across the first 200 rows, and the file size of the video in megabytes.

The frame rate estimate is computed directly from the recorded timestamps rather than from a target value, which means it reflects the actual achieved rate rather than the configured one:

```
diffs = [timestamps[i+1] - timestamps[i]
         for i in range(len(timestamps)-1)
         if timestamps[i+1] > timestamps[i]]

median_dt = sorted(diffs)[len(diffs) // 2]
fps = 1000 / median_dt if median_dt > 0 else 0
```

This verification step gives the teacher immediate confirmation that the recording completed successfully before leaving the classroom and provides the researcher with a quick sanity check on data completeness without opening any files manually (Raca & Dillenbourg, 2013).

4.7.6. Live Tracking Tab

The Live Tracking tab is owned and populated by `GazeAnalysisApp` in embedded mode. It provides a miniature canvas visualization showing the live gaze position as a red dot on a 320 × 180 pixel representation of the screen, scaled from the 1920 × 1200 coordinate space. Below the canvas, labels display the current gaze coordinates, the on-screen or off-screen status, cumulative on-screen and off-screen time percentages updated in real time, a session timer, and an optional countdown.

The FPS selector dropdown in this tab allows the teacher to change the gaze sampling rate when running in standalone eye-only mode. In synchronized dual-sensor mode, this control has no effect because `MainController` always passes `target_fps=30` to `start_recording()`, locking the rate to match the body tracking pipeline (Tobii AB, 2021).

4.7.7. Data Analysis Tab

The Data Analysis tab is populated by `GazeAnalysisApp` in embedded mode and provides the complete post-session analysis interface within the main application window. It is

organized into two functional panels: a single-session analysis panel and a multi-session comparison panel.

Single-session analysis panel. A folder path entry and a Browse button allow the teacher or researcher to navigate to any previously recorded session folder. A "Use Last Session" button populates the path automatically with the most recently completed session, stored as `root._last_session_folder` by `MainController` at the moment recording stops. Before running analysis, the panel checks that the required files `tobii_gaze.csv` and `upper_body.csv` are present in the selected folder. If either is missing, a warning dialog is shown and the analysis does not proceed.

The Run Full Analysis button triggers `engagement_analysis.run_analysis()` in a background daemon thread, preventing the GUI from freezing during the computation. A status label updates to "Running analysis… please wait" while the thread is active. When analysis completes, the background thread posts the result back to the Tkinter main thread using `root.after(0, callback)`, which is the standard pattern for safely updating widgets from a non-main thread (Lundh, 1999; Shipman, 2013):

```
def _worker():
    summary, merged = mod.run_analysis(abs_folder)
    self.root.after(0, lambda: self._on_analysis_done(
        abs_folder, summary, merged))

threading.Thread(target=_worker, daemon=True).start()
```

Once analysis completes, the results panel displays key session statistics inline: session duration, total frames processed, effective frame rate, per-signal percentage time values, overall engaged and disengaged percentages, number of disengagement events, average disengagement duration, and the mean, minimum, and maximum smoothed engagement score. The engagement score chart is displayed automatically below the statistics.

Four individual plot buttons allow the teacher to display any of the four analysis visualizations inline within the results panel: Timeline, Heatmap, Signal Summary, and Engagement Score. Each button calls `_show_analysis_image()`, which locates the corresponding PNG file in the session's `analysis/` subfolder and renders it using `Pillow` inside the results frame. This allows each visualization to be inspected interactively without opening an external image viewer.

A Download Session Report button triggers `session_exporter.export_session_xlsx()` in a background thread. The teacher is presented with a standard file-save dialog to choose the output path. Once the workbook is saved, an option to open it immediately is shown via a confirmation dialog, which uses `os.startfile()` on Windows to open the file in the default spreadsheet application:

```
if platform.system() == 'Windows':
    os.startfile(path)
```

Multi-session comparison panel. A separate panel below the single-session controls provides the multi-session comparison interface. Sessions are added individually using an Add Session button, which opens a directory browser and validates that the selected folder has already been analyzed by checking for `analysis/session_summary.json`. If a session has not yet been analyzed, the teacher is directed to run the full analysis first. Added sessions are displayed in a listbox and can be cleared with a Clear All button.

Once at least two sessions have been added, the Run Comparison button triggers `multi_session_comparison.run_comparison()` in a background thread. The comparison outputs are saved to a `comparison/` subfolder adjacent to the session folders. Three individual buttons allow the teacher to view the comparison plots inline: Score Bars, Score Curves, and Signal Breakdown. The comparison status label updates progressively to confirm how many sessions have been added and whether the minimum of two required sessions has been reached, providing clear feedback throughout the process (Goldberg et al., 2021; Prieto et al., 2016).

4.8. Implementation of Data Storage and Session Organization

4.8.1. Session Folder Creation and Naming

At the moment a recording session is confirmed by the teacher, `MainController` creates a dedicated output folder before any sensor pipeline is started. The folder is created by `_make_session_folder()`, which constructs the path from a mode prefix and a local timestamp:

```
def _make_session_folder(self, mode: str) -> str:
    ts = datetime.now().strftime('%Y%m%d_%H%M%S')
    folder = os.path.join('recordings', f"{mode}_{ts}")
    os.makedirs(folder, exist_ok=True)
    self.session_folder = folder
    return folder
```

The mode prefix encodes the sensor configuration used: `both` for synchronized dual-sensor recording, `eye` for gaze-only, and `body` for body-tracking-only. The timestamp component uses the local system time at session creation in the format `YYYYMMDD_HHMMSS`. A representative folder name for a dual-sensor session started on 18 March 2026 at 14:30:22 would therefore be `recordings/both_20260318_143022`.

This naming convention serves two practical purposes. The mode prefix makes the sensor configuration immediately identifiable when browsing the recordings directory without opening any file. The timestamp guarantees uniqueness across sessions, as two sessions cannot be started within the same second. The folder path is stored in `self.session_folder` and exposed to `GazeAnalysisApp` via `root._last_session_folder`, allowing the analysis

tab to locate the most recent session automatically after recording stops (Raca & Dillenbourg, 2013).

4.8.2. Raw Output Files

Each active recording session produces five output files written directly to the session folder by the sensor modules and the GUI. Table 11 summarizes these files, their source, sampling rate, and content.

Table 11 - Raw output files generated during each recording session, including their source modules, sampling rates, and recorded contents.

File	Source	Rate	Contents
`tobii_gaze.csv`	`GazeAppAlpha.py`	30 FPS (sync) / up to 90 FPS (standalone)	`epoch_ms`, `x`, `y`, `on_screen`
`upper_body.csv`	`body-tracker.py`	30 FPS	`epoch_ms`, `device_ms`, nose and shoulder coordinates and depths, `Distance_OK`, `Facing_Forward`, `Body_Engaged`
`hands_data.csv`	`body-tracker.py`	30 FPS	`epoch_ms`, `device_ms`, `Fingers_Detected`, `Hand_Count`
`realsense_color.avi`	`body-tracker.py`	30 FPS	Annotated RGB video, 1280 × 720, XVID codec
`session_metadata.json`	`MainApp.py`	—	Student ID, task name, facilitator, notes, ISO timestamp

Each file is written independently by the module responsible for it. This modality-specific separation means that any individual file can be re-analyzed, replaced, or inspected without affecting the others. It also means that if a sensor fails mid-session, the data from the remaining sensor remains valid and usable. All three CSV files share the same `epoch_ms` timestamp reference, ensuring that any row in one file can be cross-referenced with rows in any other file (Alkabbany et al., 2023; Raca & Dillenbourg, 2013).

4.8.3. Session Metadata File

The session metadata file `session_metadata.json` is written by `_save_session_metadata()` immediately after the teacher confirms the session details dialog, before any sensor pipeline starts. It contains the following fields:

```
{
  "student_id":  "S001",
  "task_name":   "Sprint 3 Robot Navigation",
  "facilitator": "Dr. Smith",
  "notes":       "Student arrived late, session started 5 min delayed",
  "guest":       false,
  "recorded_at": "2026-03-18T14:30:22.451823"
}
```

The `recorded_at` field is set to `datetime.now().isoformat()` at the moment the dialog is confirmed, providing a human-readable absolute timestamp for the session independent of the folder name. The `guest` boolean flag distinguishes sessions recorded with full metadata from demo or test sessions started via the Guest / Skip path, allowing analysis scripts to filter out test recordings when processing a batch of sessions.

Writing the metadata file before the sensors start ensures that if a session is terminated unexpectedly during recording, the metadata is always present in the folder. This makes it possible to identify and describe any session folder, even one that contains incomplete sensor data (Stodden et al., 2016).

4.8.4. Separation of Raw Data and Analysis Outputs

A deliberate separation is maintained between the raw sensor data files written during recording and the derived analysis outputs produced after the session. Raw files are written directly to the session folder root. All outputs produced by `engagement_analysis.py`, including the merged dataset, engagement scores, disengagement event tables, and visualization plots, are written to a dedicated `analysis/` subfolder created at analysis time:

```
out_dir = os.path.join(folder, 'analysis')
os.makedirs(out_dir, exist_ok=True)
```

The complete set of analysis outputs written to this subfolder is shown in Table 12.

Table 12 - Summary of analysis outputs generated from recorded session data, including derived metrics and visualization artifacts.

File	Description
`session_summary.json`	All session metrics in machine-readable format
`engagement_merged.csv`	Per-frame merged gaze and body data with engagement verdict
`engagement_score.csv`	Per-second engagement score, smoothed score, and signal means
`disengagement_events.csv`	Each disengagement episode with start time, end time, and duration
`engagement_timeline.png`	Four-row color band timeline: gaze, distance, facing, overall verdict
`gaze_heatmap.png`	Gaussian-smoothed gaze density map on a 1920 × 1200 canvas
`signal_summary.png`	Bar chart of percentage time each signal condition was true
`engagement_score.png`	Smoothed score over time with zone shading and individual signal traces
`session_report.xlsx`	Teacher-facing Excel workbook with five structured sheets

This separation between raw data and derived results is important for both research reproducibility and practical flexibility. Because the raw CSV files are never modified by the analysis pipeline, the analysis can be re-run at any point with updated logic, different parameters, or a corrected implementation, while the original recordings remain intact. This property is a fundamental requirement for research validity, as observations and their interpretation must remain separable (Romero & Ventura, 2010; Stodden et al., 2016). It also allows multiple analysis runs to coexist in the same session folder if needed, since the `analysis/` subfolder can be deleted and regenerated without affecting the source data.

4.8.5. File Handle Management and Data Integrity

File handles for all CSV files and the video writer are kept open continuously from the moment recording starts until `stop_recording()` is called. This avoids the overhead of repeatedly opening and closing files on each write, which would be significant at 30 FPS. Both the gaze CSV writer in `GazeAppAlpha.py` and the body CSV writers in `body-tracker.py` flush their buffers periodically during recording, every 30 rows for the gaze module and after every row for the body module, providing protection against data loss from unexpected process termination (Gorelick & Ozsvald, 2020).

When `stop_recording()` is called, the gaze module performs an explicit flush and close of the CSV file handle before returning:

```
def stop_recording(self):
    if not self._recording:
        return
    try:
        if self._record_file:
            self._record_file.flush()
            self._record_file.close()
    except Exception:
        pass
    self._record_file = None
    self._record_writer = None
    self._recording = False
```

The body tracking module performs the equivalent operation in `SessionLogger.close()`, with the additional step of acquiring the `threading.Lock` before calling `VideoWriter.release()` to prevent a race condition with the preview thread. This ordered shutdown sequence ensures that all file handles are cleanly closed and all buffered data is written to disk before the post-recording verification panel reads the files (Martelli et al., 2005).

4.9. Implementation of Engagement Analysis and Reporting

4.9.1. Analysis Pipeline Overview

The post-session engagement analysis is implemented in `engagement_analysis.py` as a self-contained pipeline invoked through a single public function, `run_analysis(folder)`. The function accepts the path to a session recording folder and executes the following sequence: load the session CSV files, merge the two data streams by timestamp, compute the engagement verdict and Focus Index for every frame, aggregate summary statistics, detect disengagement events, and save all derived outputs to the session's `analysis/` subfolder. It returns a `(summary, merged)` tuple so that the GUI can display key statistics immediately after analysis completes without a secondary file read.

The pipeline is designed to be fully deterministic given the same input files. Because all analysis is post-session and reads only from the raw CSV files, re-running the analysis on the same session always produces identical outputs, which is a prerequisite for research reproducibility (Stodden et al., 2016).

4.9.2. Loading and Preparing Session Data

The `load_session()` function reads the three CSV files from the session folder. Both `tobii_gaze.csv` and `upper_body.csv` are required, while `hands_data.csv` is treated as optional and its absence does not prevent the analysis from running. Each file is loaded

into a pandas DataFrame and sorted by `epoch_ms` to ensure chronological ordering before any merge operation:

```
df = pd.read_csv(path)
df = df.sort_values('epoch_ms').reset_index(drop=True)
```

Explicit sorting is necessary because, although both pipelines write rows in chronological order during recording, rows are not guaranteed to be perfectly ordered if any write operation was retried or if the file system reordered buffered writes. Sorting by `epoch_ms` before merging eliminates any such ambiguity (McKinney, 2010).

4.9.3. Stream Merging and Engagement Verdict

The `build_merged()` function performs the nearest-neighbour temporal merge described in Section 4.6, using the body stream as the anchor and joining the closest gaze sample to each body frame within the 50 ms tolerance. After the merge, timestamps are normalized to seconds from the session start by computing `t = (epoch_ms – t0) / 1000.0`, where `t0` is the `epoch_ms` value of the first body frame. This produces a relative time axis used throughout all subsequent analysis and visualization.

The three-signal engagement verdict is then computed as a Boolean column on the merged DataFrame. A student is classified as disengaged only when all three signals fail simultaneously, meaning gaze is off-screen, distance is outside threshold, and the student is not facing forward:

```
merged['Engaged'] = ~(
    (~merged['on_screen']) &
    (~merged['Distance_OK']) &
    (~merged['Facing_Forward'])
)
```

This conservative rule ensures that legitimate off-screen activity, such as working with a robot, does not generate a false disengagement classification. The rationale and pedagogical justification for this rule are established in the Design Phase, and the implementation here directly realizes that specification (Booth et al., 2023; Bosch et al., 2015).

4.9.4. Focus Index Computation

The Focus Index quantifies screen directed behavioral engagement, defined as the degree to which a student's observable actions are oriented toward the workstation. It is computed by the compute_engagement_score() function in three stages. As defined in the Design

Phase, the metric is not intended to capture task engagement during robot interaction phases, where gaze and body are appropriately directed away from the screen.

In the first stage, a raw weighted score is computed for every frame using the formula defined in the Design Phase:

```
df['raw_score'] = (
    df['on_screen'].astype(float)       * W_GAZE   +
    df['Facing_Forward'].astype(float)  * W_FACING +
    df['Distance_OK'].astype(float)     * W_DIST
) * 100
```

The weight constants are `W_GAZE = 0.50`, `W_FACING = 0.30`, and `W_DIST = 0.20`, reflecting the relative importance of each signal as an indicator of student engagement in the educational robotics context (Booth et al., 2023; Zaletelj & Košir, 2017).

In the second stage, frames are grouped into one-second intervals and the mean raw score for each second is computed:

```
df['t_bin'] = df['t'].apply(lambda t: int(t))
per_second = df.groupby('t_bin').agg(
    score    = ('raw_score',      'mean'),
    gaze     = ('on_screen',      'mean'),
    facing   = ('Facing_Forward', 'mean'),
    distance = ('Distance_OK',    'mean'),
).reset_index()
```

This per-second aggregation reduces the influence of momentary single-frame fluctuations and produces a more stable representation of engagement trends. The use of one-second aggregation windows for engagement scoring is consistent with established practice in automated classroom monitoring (Alkabbany et al., 2023; Sümer et al., 2021).

In the third stage, a centered three-second rolling average is applied to the per-second scores to generate the final smoothed Focus Index:

```
per_second['score_smooth'] = (
    per_second['score']
    .rolling(window=SCORE_SMOOTH_S, center=True, min_periods=1)
    .mean()
)
```

The constant `SCORE_SMOOTH_S = 3` defines a three-second smoothing window. Using a centered window means that each smoothed value reflects one second before and one second after the current second, which prevents the smoothed score from lagging behind the actual engagement state. The `min_periods=1` parameter ensures that the first and last seconds of the session receive valid smoothed scores rather than being set to NaN (McKinney, 2010). The resulting smoothed score is interpreted across three engagement zones: high (≥ 80), moderate (50–79), and low (< 50).

4.9.5. Disengagement Event Detection

Disengagement events are identified by `compute_summary()` as contiguous runs of frames where the `Engaged` column is `False`. The detection iterates through the merged DataFrame and records the start and end time of each run:

```
for _, row in merged.iterrows():
    if not in_event and not row['Engaged']:
        in_event = True
        start_t = row['t']
    elif in_event and row['Engaged']:
        events.append({'start_s': round(start_t, 2),
                       'end_s':   round(row['t'], 2),
                       'duration_s': round(row['t'] - start_t, 2)})
        in_event = False
```

An open event at the end of the session, where the student is disengaged when recording stops, is correctly closed against the session duration. For each event, the duration in seconds is recorded alongside the absolute start and end times, enabling both per-event analysis and aggregate statistics. The average disengagement duration across all events is computed and included in the session summary (Dewan et al., 2019).

4.9.6. Visualization Outputs

Four analysis plots are generated by dedicated functions and saved as PNG files in the `analysis/` subfolder at 150 DPI.

`plot_engagement_timeline()` produces a stacked four-row color band plot showing `on_screen`, `Distance_OK`, `Facing_Forward`, and the overall `Engaged` verdict as binary time series across the full session duration. This visualization allows a researcher to inspect the temporal pattern of each signal independently and identify moments where multiple signals fail simultaneously.

`plot_gaze_heatmap()` constructs a spatial density map of on-screen gaze positions. Only samples with `on_screen = True` and valid coordinates are included. The coordinates are scaled to the 1920 × 1200 pixel screen space and accumulated into a density grid. A Gaussian blur with `HEATMAP_SIGMA = 30` pixels is applied using `scipy.ndimage.gaussian_filter` to smooth the distribution and improve interpretability:

```
heatmap = np.zeros((SCREEN_H, SCREEN_W), dtype=np.float32)
for x, y in zip(xs, ys):
    heatmap[y, x] += 1
heatmap = gaussian_filter(heatmap, sigma=HEATMAP_SIGMA)
```

The resulting image is rendered using a jet colormap on a black background, with warmer colors indicating higher gaze density. This is a standard representation for gaze density visualization in eye-tracking research (Appel et al., 2018; Tobii AB, 2021).

`plot_signal_summary()` produces a bar chart showing the percentage of session time each signal condition was true, including gaze on-screen, distance OK, facing forward, body engaged, and overall engaged. This provides a compact overview of the session that is accessible to non-technical users.

`plot_engagement_score()` produces a line chart of the smoothed Focus Index over time, with background color bands indicating the three engagement zones. Individual signal traces are overlaid as thin transparent lines, and a horizontal dashed line marks the session mean score.

4.9.7. Excel Teacher Report

`session_exporter.py` generates a structured Excel workbook for teacher-facing reporting using `openpyxl`. The workbook contains five sheets, each serving a distinct purpose:

Session Report: summary of session metadata, duration, engagement statistics, disengagement events, and signal weights

Engagement Summary: one row per second showing score values and signal means

Gaze Data: full `tobii_gaze.csv` content

Body Data: merged body tracking data with engagement verdicts

Disengagement Events: all disengagement episodes with severity classification and interpretation

The workbook is styled using a consistent color scheme with formatted headers and structured layout for readability. Presenting analysis results in a spreadsheet format reduces the barrier for non-technical users compared to raw data formats and aligns with established recommendations for teacher-facing analytics tools (Goldberg et al., 2021; Prieto et al., 2016).

4.9.8. Multi-Session Comparison

`multi_session_comparison.py` supports longitudinal analysis by accepting two or more analyzed session folders and producing side-by-side comparison outputs. It loads `session_summary.json` and `engagement_score.csv` from each session's `analysis/` subfolder and generates three plots and one summary table:

`comparison_scores.png`: bar chart of mean engagement scores per session

`comparison_signals.png`: grouped bar chart comparing signal percentages

`comparison_curves.png`: overlaid engagement score curves across sessions

`comparison_table.csv`: summary table with key metrics per session

This multi-session capability supports the broader research objective of the ALPER-EU project, where engagement patterns across multiple students and classroom activities are analyzed over time (Booth et al., 2023; Dewan et al., 2019).

4.10. Error Handling and Practical Robustness

4.10.1. Overview

A research prototype deployed in a real classroom environment must handle unexpected conditions gracefully without losing recorded data, crashing mid-session, or requiring technical intervention from the teacher. The implementation addresses robustness at several distinct levels: sensor signal absence, frame-level inference failure, concurrent resource access, session teardown integrity, and post-recording verification. Each of these is described in the subsections below.

4.10.2. Tobii Signal Loss and Silence Detection

The gaze acquisition module is designed to tolerate complete loss of the TobiiStream signal without crashing or corrupting the recorded data. The `tracking_loop` background thread uses non-blocking ZeroMQ polling with `socket.poll(0)`, which returns immediately if no message is available rather than blocking indefinitely. This means that if the TobiiStream service stops publishing, due to a cable disconnection, driver crash, or sensor power loss, the loop continues executing normally without freezing (Hintjens, 2013).

Signal absence is detected by the 500 millisecond silence threshold in `_normalize_gaze_data()`. When the time since the last received message exceeds this threshold, the sample is classified as off-screen regardless of the last known coordinates, preventing stale values from being recorded as valid gaze data. This behavior continues automatically for the remainder of the session, so a brief sensor disconnection followed by reconnection is handled transparently. The silence period is recorded as off-screen, and

valid gaze resumes as soon as messages are received again, without any explicit recovery logic (Tobii AB, 2021).

If the ZeroMQ context itself is terminated, for example when `shutdown()` is called at the end of one session before a new session starts, it is recreated lazily in `start_streaming_preview()`:

```
if self.context is None:
    self.context = zmq.Context()
```

This allows the gaze module to be restarted between sessions within the same application process without error (Hintjens, 2013).

4.10.3. MediaPipe Landmark Absence

The body tracking pipeline handles frames in which MediaPipe fails to detect any pose landmarks without interrupting the recording loop. The `_preview_loop` passes `pose_results.pose_landmarks` if `pose_results` else `None` to `log_frame()`. Inside `log_frame()`, the CSV write for `upper_body.csv` is guarded by a check on whether landmarks were detected:

```
if pose_landmarks:
    # write landmark data and engagement signals
    ...
```

When `pose_landmarks` is `None`, which occurs when the student is partially outside the camera's field of view or when MediaPipe temporarily fails to detect a pose, no row is written to `upper_body.csv` for that frame. The video frame is still written, and the hands CSV row is still written, so the other outputs remain continuous. The resulting gap in `upper_body.csv` is handled correctly during post-session analysis by the nearest-neighbour merge tolerance, which treats the missing body frames as periods with no body signal (McKinney, 2010).

If `log_frame()` itself raises an exception, for example due to a file handle error, it is caught at the call site and logged as a warning without propagating:

```
try:
    self.logger.log_frame(...)
except Exception as e:
    print(f"[WARN] logger.log_frame failed: {e}")
```

This ensures that a single failed write does not terminate the preview loop or corrupt subsequent frames (Gorelick & Ozsvald, 2020).

Similarly, individual CSV write operations inside `log_frame()` are wrapped in their own try-except blocks, so a failure to write one row does not prevent subsequent rows from being written:

```
try:
    self.upper_writer.writerow(row)
    self.upper_f.flush()
except Exception as e:
    print(f"[WARN] upper_body CSV write failed: {e}")
```

4.10.4. Body Tracker Preview Loop Resilience

The entire body tracker preview loop runs inside a `try/finally` block. Regardless of how the loop exits, whether through the normal `_stop_event` signal, an unhandled exception, or the ESC key, the `finally` clause guarantees that the RealSense pipeline is stopped, the MediaPipe models are closed, all OpenCV windows are destroyed, and the `_preview_running` flag is reset to `False`:

```
finally:
    if self.logger:
        try:
            self.logger.close()
        except Exception:
            pass
    for resource, name in ((pipeline, "pipeline"), (pose, "pose"),
                           (hands_model, "hands")):
        try:
            resource.stop() if name == "pipeline" else resource.close()
        except Exception:
            pass
    cv2.destroyAllWindows()
    self._preview_running = False
```

Each resource release is wrapped in its own try-except block so that a failure to close one resource does not prevent the others from being released. This ordered teardown pattern is standard practice for reliable resource management in Python applications that hold multiple hardware handles (Beazley & Jones, 2013; Martelli et al., 2005).

4.10.5. Duplicate Recording Guard

If `start_recording()` is called on the gaze module while a recording is already open, for example if a session is started twice without stopping, the existing recording is closed cleanly before the new one is opened, rather than silently ignoring the call or leaving an orphaned file handle:

```
if self._recording:
```

```
    self.stop_recording()
```

This guard prevents a class of bugs in which a previous session's CSV file handle is leaked, causing data from the new session to be appended to the old file or the old file to remain locked on the filesystem (Python Software Foundation, 2024).

4.10.6. GUI Shutdown Sequence

When the application window is closed, the `on_closing()` handler is registered with Tkinter's `WM_DELETE_WINDOW` protocol. This handler calls `gaze_app.shutdown()` before destroying the window, ensuring that the ZeroMQ context is terminated, the tracking thread is joined, and all pending `root.after()` callbacks are cancelled before the process exits:

```
def shutdown(self):
    self._shutdown = True
    self.is_tracking = False
    if self.tracking_thread and self.tracking_thread.is_alive():
        self.tracking_thread.join(timeout=1.0)
    if self.socket:
        self.socket.close()
        self.socket = None
    if self.context:
        self.context.term()
        self.context = None
```

The `_shutdown` flag prevents the `process_queue()` GUI callback from rescheduling itself after the window has been destroyed, which would otherwise cause a `TclError` when the callback attempts to update a widget that no longer exists (Shipman, 2013). Without this mechanism, closing the application while tracking is active would result in a runtime error or a hang while waiting for the background thread.

4.10.7. Post-Recording Data Verification

After every session stops, `verify_recording()` runs automatically and performs a lightweight integrity check on all four output files. For each of the three CSV files, it reads the first 200 rows, counts the total row count, and estimates the actual achieved frame rate from the median inter-timestamp difference:

```
diffs = [timestamps[i+1] - timestamps[i]
         for i in range(len(timestamps)-1)
         if timestamps[i+1] > timestamps[i]]
median_dt = sorted(diffs)[len(diffs) // 2]
fps = 1000 / median_dt if median_dt > 0 else 0
```

For the video file, it checks that the file exists and that its size is greater than 0.01 MB, a threshold chosen to detect the degenerate case where the file was created but no frames

were written. The results are displayed in the verification panel with clear pass (✓), warning (⚠), or failure (✗) indicators. This gives the teacher immediate confirmation before leaving the classroom that the session was recorded successfully, and gives the researcher a quick check on data completeness without requiring manual file inspection (Raca & Dillenbourg, 2013).

4.11. Sensor Interface Requirements and Extensibility

The system was designed with sensor decoupling as a deliberate architectural principle. The gaze acquisition module does not communicate directly with the Tobii Eye Tracker 4C hardware, and the body tracking pipeline operates entirely on standard NumPy arrays and OpenCV frames beyond the RealSense initialization block. This section defines the interface contracts expected by both sensing pipelines, thereby specifying the requirements that any replacement device must satisfy at the code level.

4.11.1. Gaze Interface Contract

The gaze module in `GazeAppAlpha.py` subscribes to a ZeroMQ PUB socket at `tcp://127.0.0.1:5556` and expects messages in the following space-delimited string format:

```
TobiiStream <sequence> <raw_x> <raw_y> <validity>
```

where `raw_x` and `raw_y` are floating point gaze coordinates expressed in screen pixels, and `validity` is an integer quality flag.

The module has no knowledge of the underlying hardware. Any eye tracker that can be bridged to publish messages in this format over the same socket is fully compatible with the system without requiring any modification to `GazeAppAlpha.py`.

The on-screen classification bounds are defined by the constants `SCREEN_W = 1920` and `SCREEN_H = 1200` in `engagement_analysis.py`, together with the normalization bounds implemented in `_normalize_gaze_data()` in `GazeAppAlpha.py`. These values must be updated if the display resolution differs.

4.11.2. Body Tracking Interface Contract

The body tracking pipeline in `body-tracker.py` uses `pyrealsense2` only for stream initialization and frame acquisition. Once frames are retrieved, all subsequent processing operates on standard data types and is hardware independent.

A replacement depth camera must provide the following outputs for each frame:

- A `color_image` represented as a NumPy array of shape `(H, W, 3)` in BGR format at 30 FPS
- A depth lookup callable equivalent to `depth_frame.get_distance(x, y)` returning the distance in metres for a given pixel coordinate

The RealSense specific code is confined to the initialization block and frame retrieval loop. All downstream processing, including MediaPipe pose inference, distance computation, orientation detection, CSV writing, and video recording, is independent of the underlying hardware and requires no modification when substituting a different depth camera.

For alternative Intel RealSense models such as the D435 or D415, no code changes are required, as these devices use the same `pyrealsense2` API with identical stream configuration calls.

4.12. Implementation Phase Summary

This chapter has presented the practical realization of the ALPER-EU Multimodal Engagement Assessment System, describing how the architecture, synchronization strategy, behavioral logic, and storage design defined in the Design Phase were translated into working software.

The system was implemented as a modular Python application organized across four primary modules. `MainApp.py` serves as the session orchestrator, coordinating sensor startup, synchronization gating, live engagement monitoring, and post-recording verification. `GazeAppAlpha.py` implements the Tobii gaze capture pipeline using a two-loop background thread architecture that separates message reception from timed sample production, ensuring that the most recent gaze position is always used regardless of message arrival rate. `body-tracker.py` implements the RealSense body tracking pipeline, combining MediaPipe pose and hand inference with depth-based distance estimation, body orientation detection through a three-cue rotation score, and synchronized XVID video recording. `engagement_analysis.py` implements the post-session analysis pipeline, producing the Focus Index, behavioral timelines, gaze heatmaps, disengagement event tables, and teacher-facing Excel reports from the recorded session data.

Temporal synchronization between the two sensing pipelines is achieved through a two-stage mechanism. A shared `go_epoch_ms` gate, computed before the session countdown begins, ensures that both CSV datasets start at the same absolute Unix millisecond

timestamp, while a nearest-neighbour merge using `pandas.merge_asof` with a 50 ms tolerance aligns individual samples during post-session analysis. Both stages rely exclusively on host-assigned timestamps, avoiding the inconsistencies that would arise from comparing independent sensor hardware clocks.

The implementation was designed with practical robustness in mind. Signal loss from the Tobii device is handled transparently through a 500 ms silence threshold. MediaPipe landmark absence produces natural gaps in the body CSV rather than errors. All file handles are flushed periodically during recording and closed in an ordered sequence on session stop. A post-recording verification panel confirms row counts, estimated frame rates, and video file size immediately after every session.

The resulting system satisfies all functional requirements established in the User Needs Analysis, including single-button session control with automatic synchronized recording, objective assessment of screen-directed behavioral engagement through a transparent rule-based Focus Index, context-aware multimodal interpretation that distinguishes robot-related activity from disengagement, millisecond-resolution temporal alignment, synchronized video for post-session validation, structured and reproducible session data storage, and minimal classroom disruption through fully automated data acquisition.

The Implementation Phase establishes the complete software foundation on which evaluation and experimental testing of the system can be conducted. A key interpretive constraint of the implemented system should be noted. The Focus Index measures screen directed behavioral engagement and reflects reduced scores during robot interaction phases regardless of actual task engagement. This is an inherent property of gaze based behavioral proxies in mixed modality learning environments and does not represent a failure of the implementation. Formal validation of the metric against observer annotated engagement labels, as well as contextual interpretation of score reductions during known robot interaction intervals, are identified as priorities for the evaluation phase.

5. Results & Evaluation

5.1. Overview

This chapter presents the results of a preliminary evaluation session conducted using the ALPER-EU Multimodal Engagement Assessment System. The session was recorded on 22 April 2026 using both sensing pipelines simultaneously, namely the Tobii Eye Tracker 4C for gaze data acquisition and the Intel RealSense D455 for body tracking and video recording.

The evaluation demonstrates the system operating under realistic conditions during a continuous one-hour session, producing synchronized multimodal data, real time engagement monitoring, and a complete set of post session analysis outputs.

The session took place in a desktop workstation environment representative of an educational robotics learning context, with the participant engaged in a code development task.

5.2. System Startup and Session Initialization

Upon application launch, the Session Control tab performs an automatic sensor readiness check approximately 1.5 seconds after startup. Both the Eye Tracker and Body Tracker indicators update to reflect the connection status of the physical devices. Figure 7 shows the application at startup, with both sensors reporting *Connected* in green, confirming that the Tobii Eye Tracker 4C is reachable via the TobiiStream bridge and that the Intel RealSense D455 has been successfully detected by the pyrealsense2 runtime.

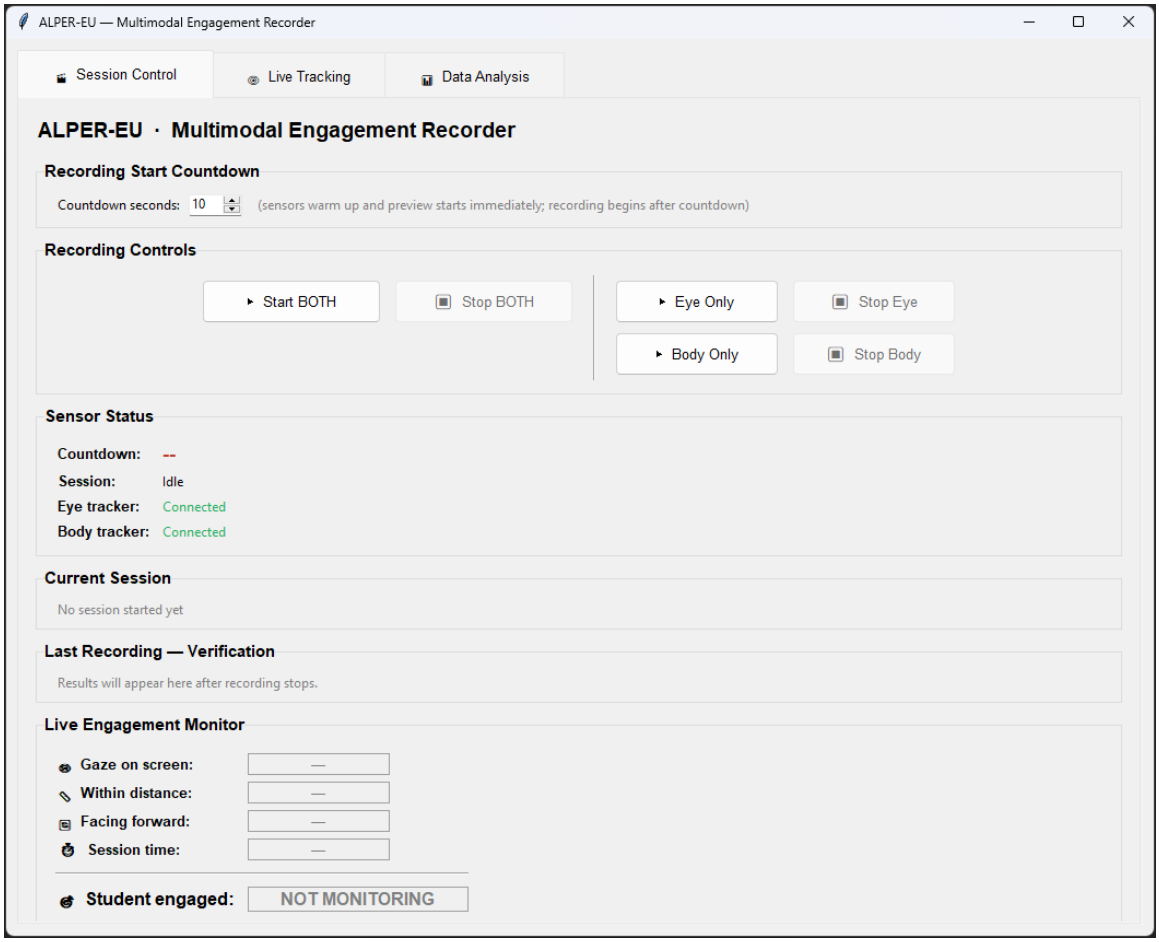


Figure 7 - The Session Control tab at application startup. Both the Eye Tracker and Body Tracker indicators show Connected, confirming that both sensing pipelines have been successfully detected prior to recording.

Before starting a recording session, the teacher completes a session metadata dialog that captures the student identifier, task name, facilitator name, and optional session notes. Figure 8 shows the completed dialog for the evaluation session used in this chapter.

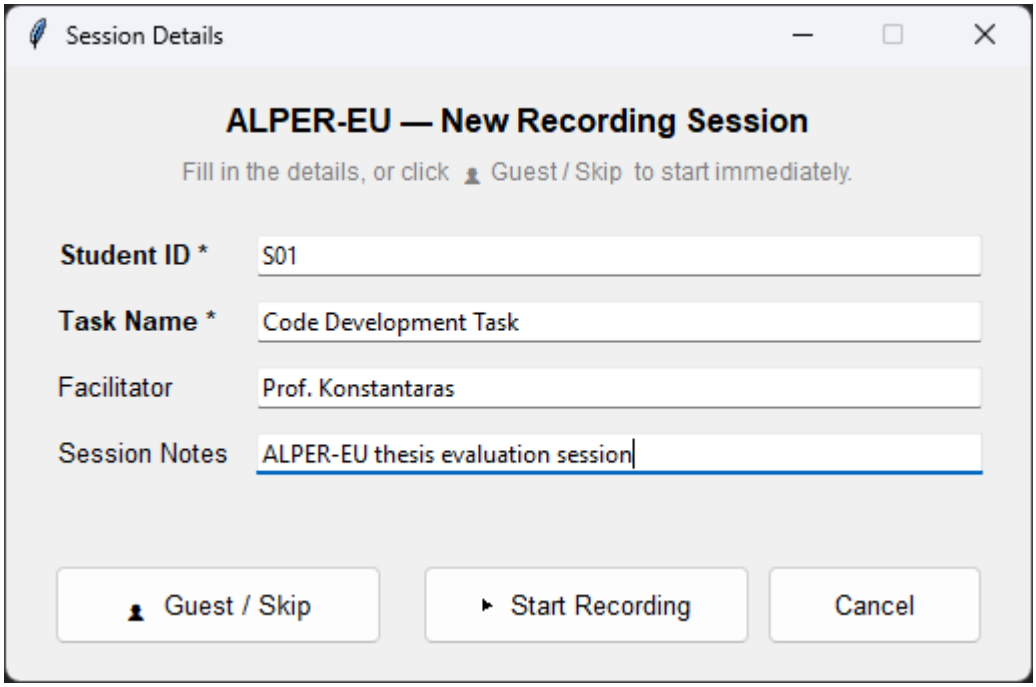


Figure 8 - The session metadata dialog completed with Student ID S01, Task Name "Code Development Task", Facilitator "Prof. Konstantaras", and Session Notes "ALPER-EU thesis evaluation session". This metadata is stored in the session_metadata.json file at the start of each recording.

After confirming the metadata, a countdown is initiated. Both sensor pipelines immediately enter preview mode, allowing the teacher to verify that the sensors are active and that the student is correctly positioned before data collection begins. Figure 9 shows the Session Control tab during the 30 second countdown period, with both pipelines reporting *Previewing*.

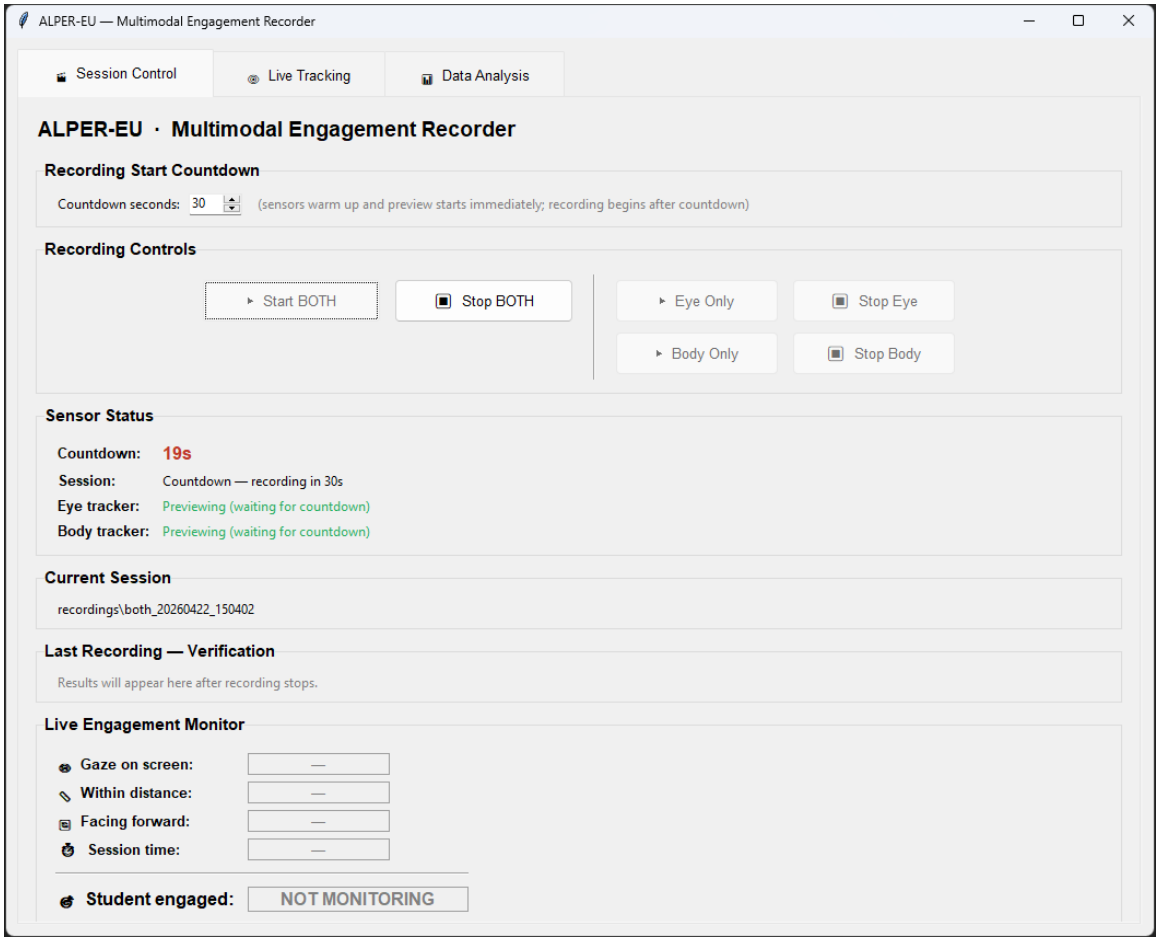


Figure 9 - The Session Control tab during the pre recording countdown. Both sensors are active in preview mode while the countdown runs, ensuring stable initialization prior to data collection.

5.3. Live Engagement Monitoring During Recording

Once the countdown expires, recording begins and the Live Engagement Monitor updates every 200 milliseconds. The monitor aggregates signals from both sensing pipelines and applies the three signal disengagement rule to produce a live engagement verdict. Figure 10 shows the Session Control tab during active recording, with all three signals reporting positively and the overall verdict indicating *ENGAGED*.

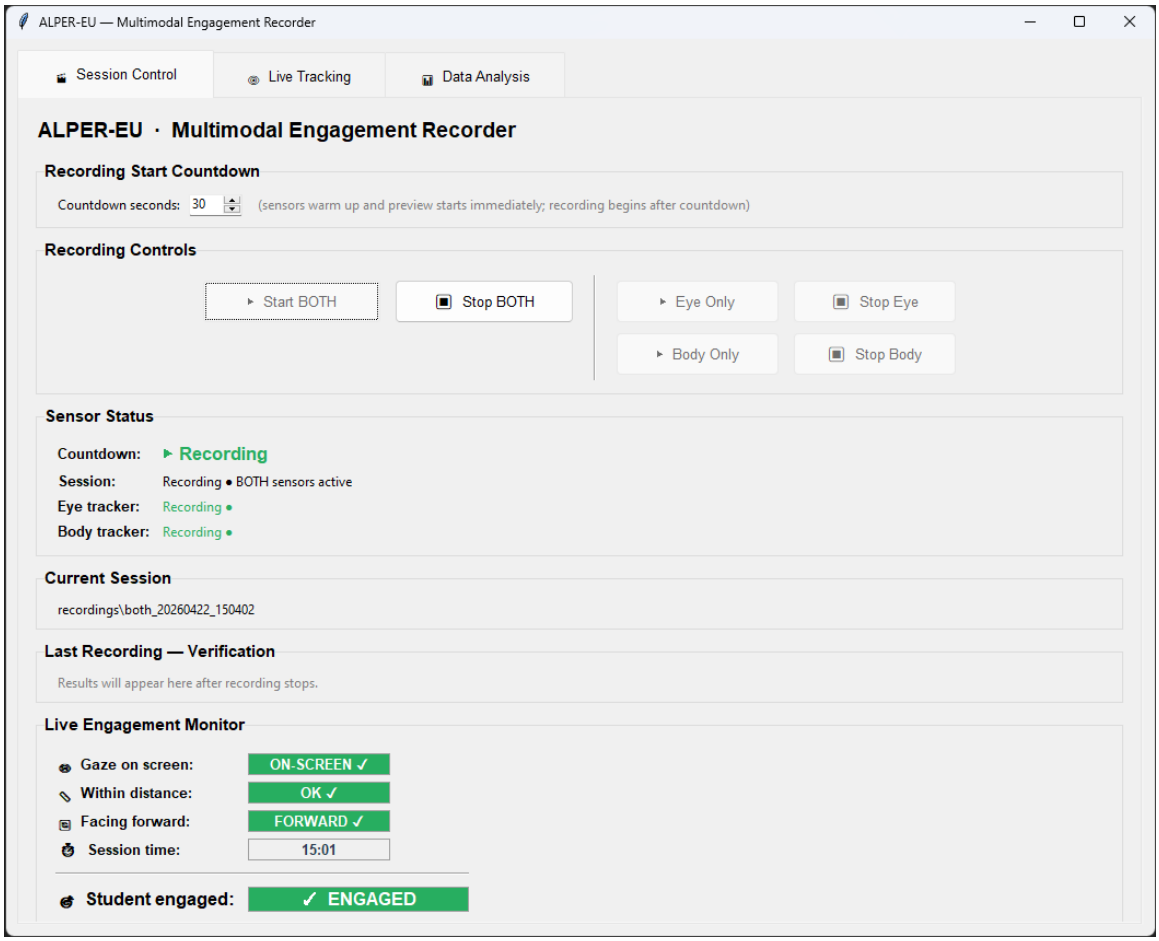


Figure 10 - The Session Control tab during active recording at 15 minutes and 1 second into the session. All three signal indicators are green — ON SCREEN ✓, OK ✓, and FORWARD ✓— and the overall verdict shows ENGAGED. Both sensing pipelines confirm Recording status.

The Live Tracking tab provides a real time visualization of the gaze stream throughout the session. Figure 11 shows the tab during active recording, displaying the current gaze position as a red dot on a miniature screen canvas alongside cumulative on screen and off screen duration statistics.

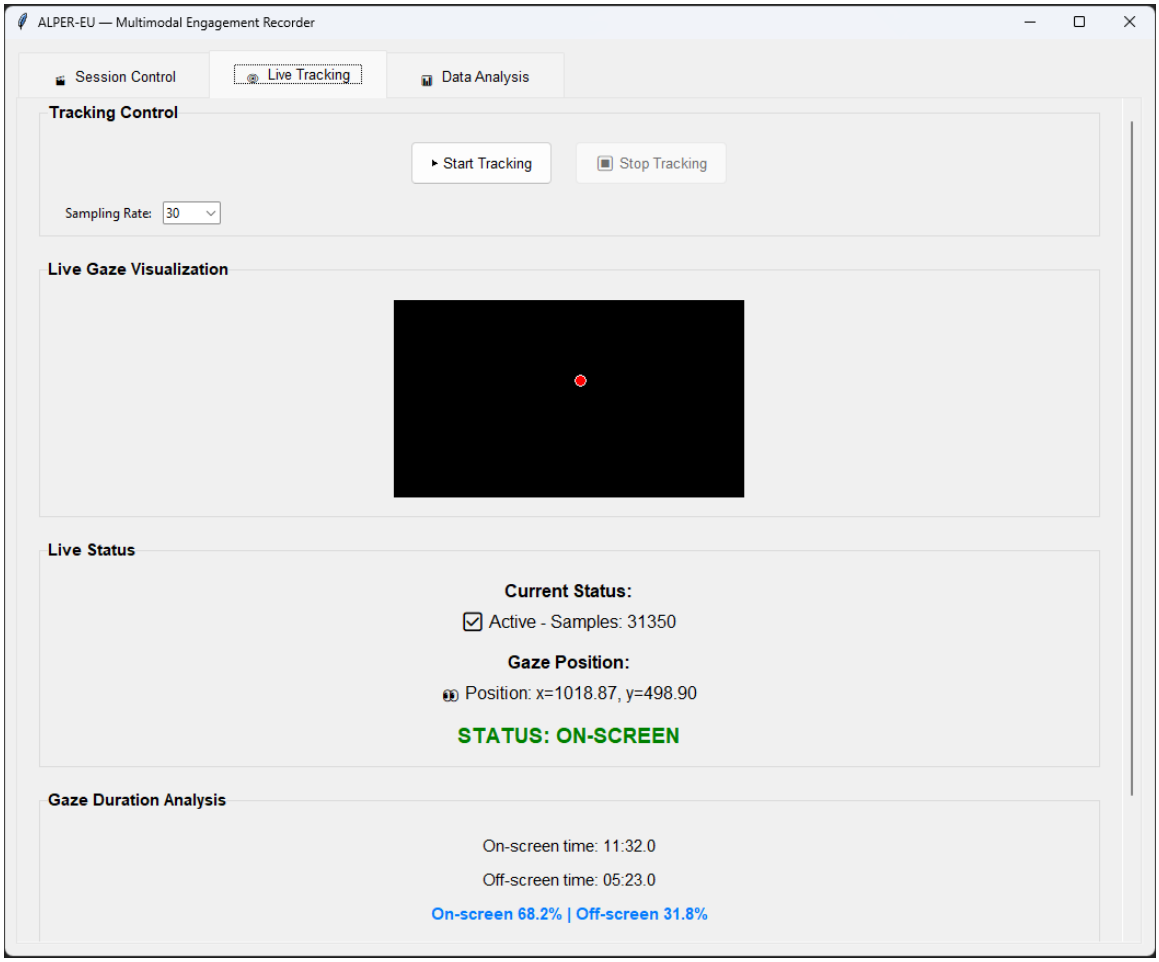


Figure 11 - The Live Tracking tab during active recording. The red dot indicates the current gaze position at screen coordinates (1018.87, 498.90). The session has accumulated 31,350 gaze samples with an on screen rate of 68.2%, reflecting natural task behaviour including brief look aways.

A key design property of the system is that the disengagement verdict requires all three signals to fail simultaneously. Figure 12 demonstrates this conservative rule in practice. Two signals are failing (gaze off screen and body turned), yet the system correctly maintains an *ENGAGED* verdict because the distance signal remains active.

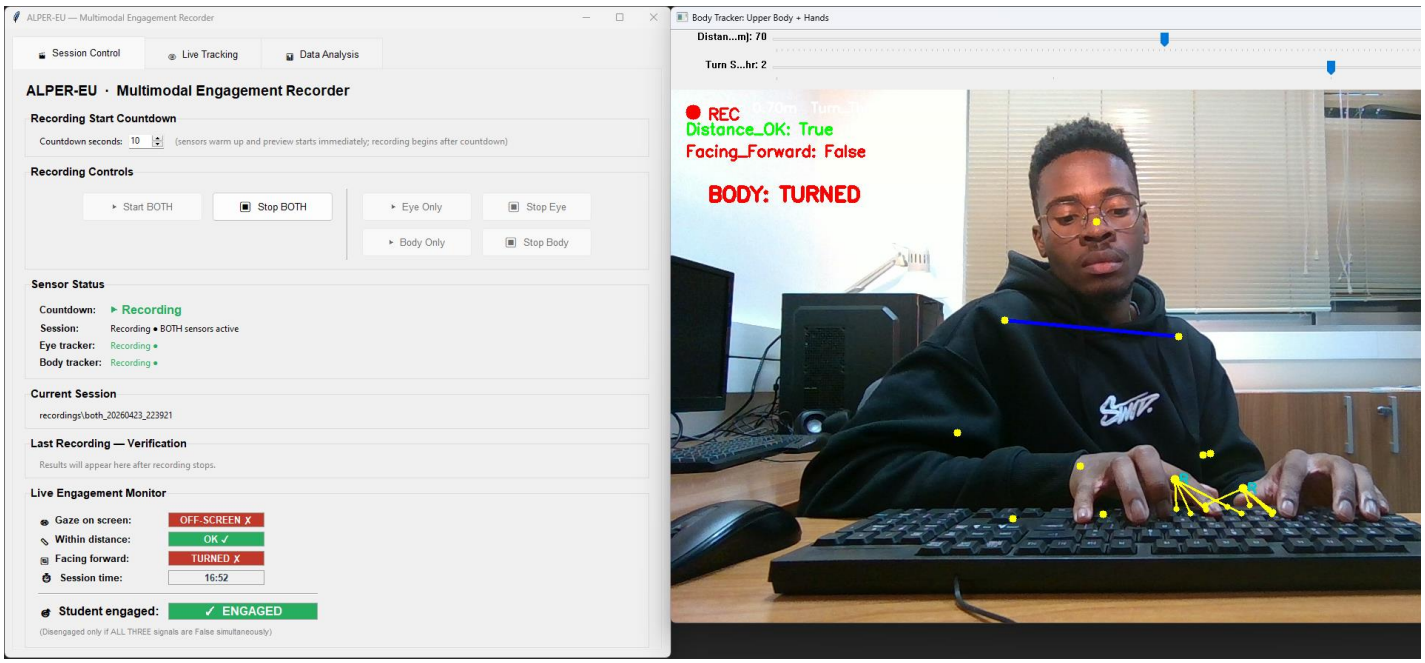


Figure 12 - The Live Engagement Monitor at session time 36:15 showing two signals failing simultaneously — OFF SCREEN ✗ and TURNED ✗ — while Distance OK remains active. The overall verdict remains ENGAGED, reflecting the three signal disengagement rule in which one or two failing signals are insufficient to classify disengagement.

Figure 13 shows the only condition under which disengagement is triggered, namely when all three signals fail simultaneously.

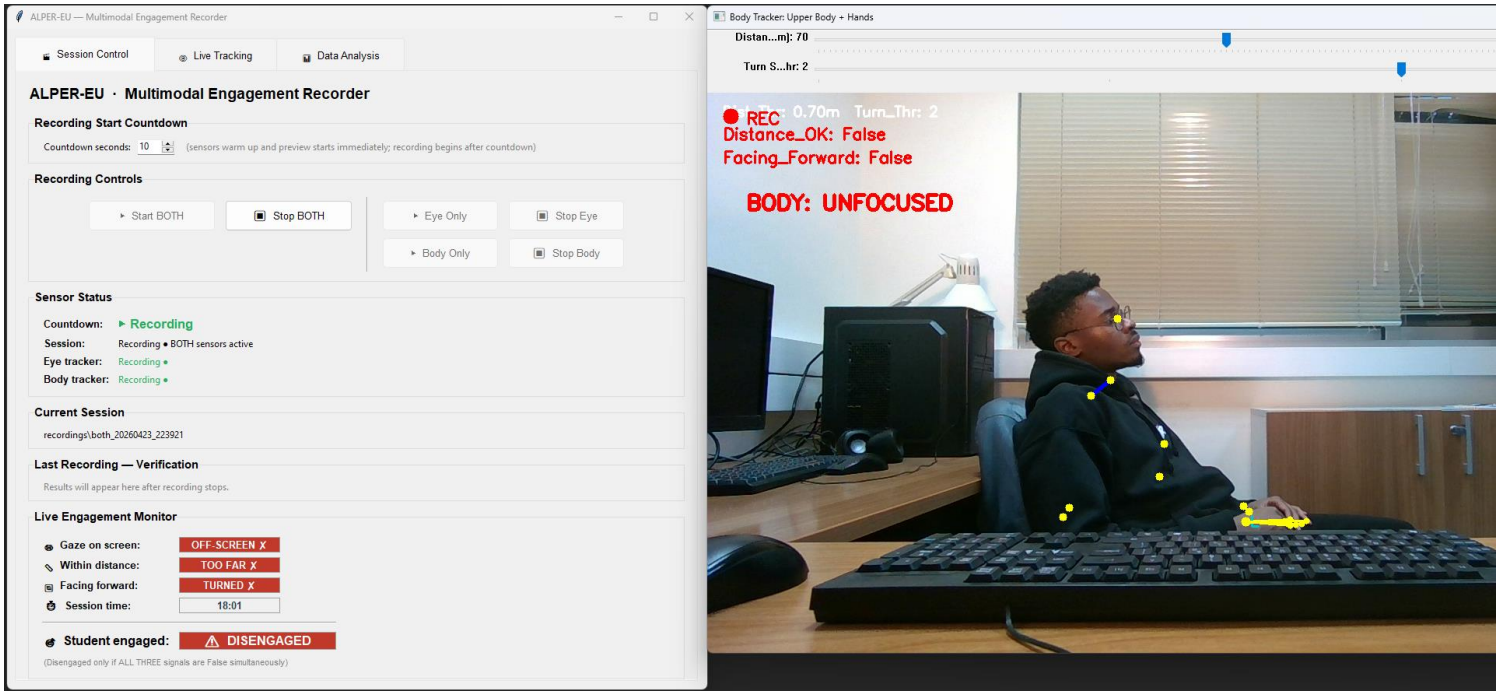


Figure 13 - The Live Engagement Monitor at session time 37:30 showing all three signals failing simultaneously — OFF SCREEN ✗, TOO FAR ✗, and TURNED ✗. The overall verdict updates to DISENGAGED, corresponding to a complete withdrawal from the workstation.

5.4. Post Recording Verification and Output Files

After the recording is stopped, the application automatically performs a post recording verification check on all output files. Figure 14 shows the verification panel confirming that all output files were written successfully.

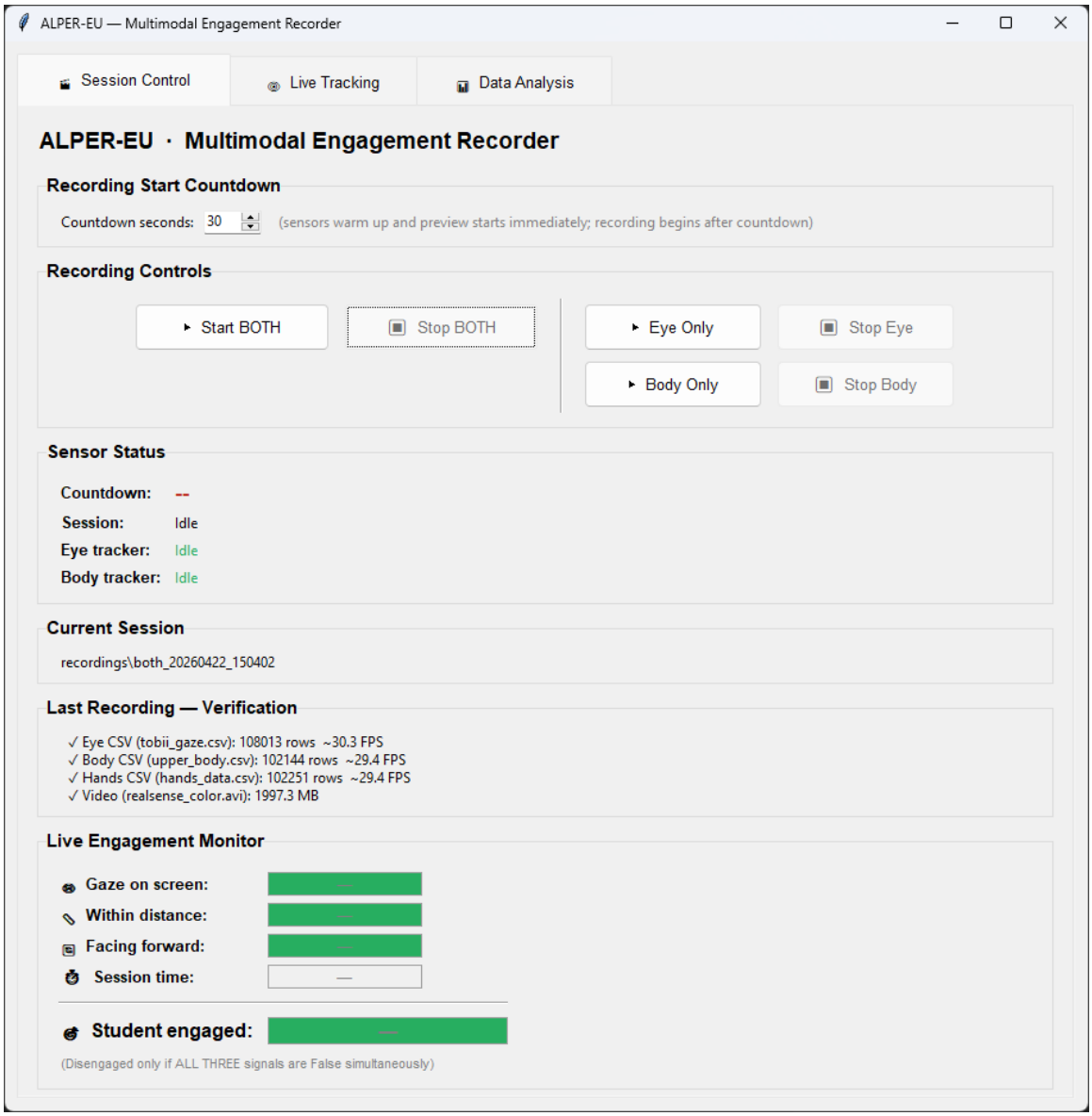


Figure 14 - The post recording verification panel confirming successful data capture. The Eye CSV contains 108,013 rows at approximately 30.3 FPS, the Body CSV contains 102,144 rows at approximately 29.4 FPS, the Hands CSV contains 102,251 rows at approximately 29.4 FPS, and the video file is 1997.3 MB. All files pass verification.

All session output files are stored in a single timestamped folder under the `recordings/` directory. Figure 15 shows the session folder contents as viewed in File Explorer.

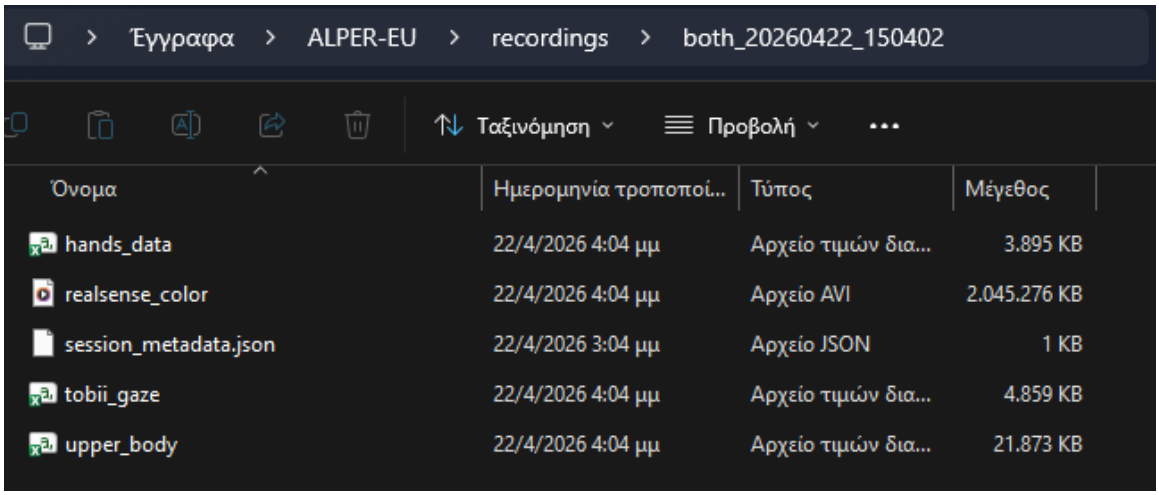
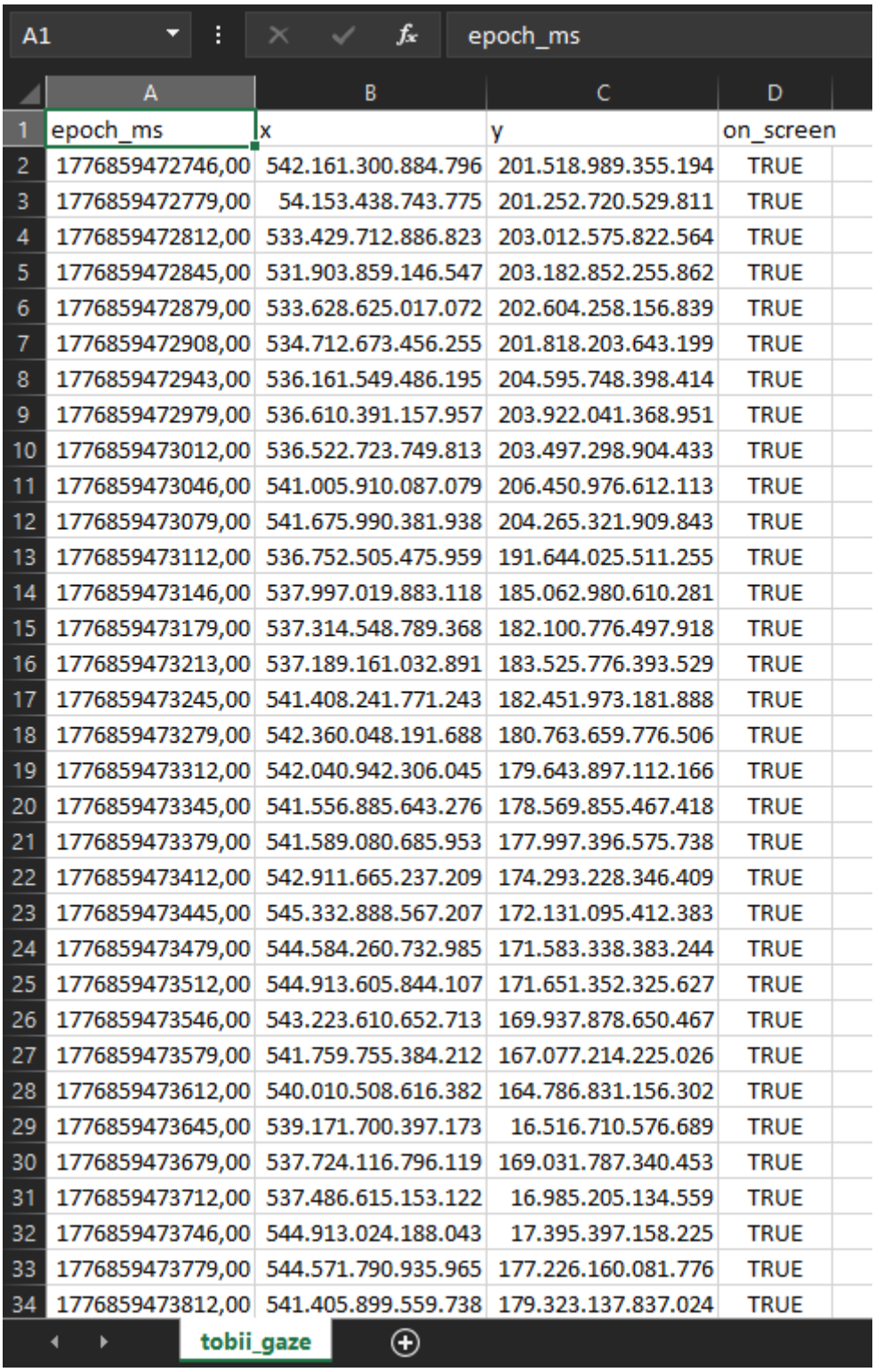


Figure 15 - The session output folder both_20260422_150402 containing all generated files: tobii_gaze.csv (4.86 MB), upper_body.csv (21.87 MB), hands_data.csv (3.90 MB), realsense_color.avi (2.00 GB), and session_metadata.json (1 KB).

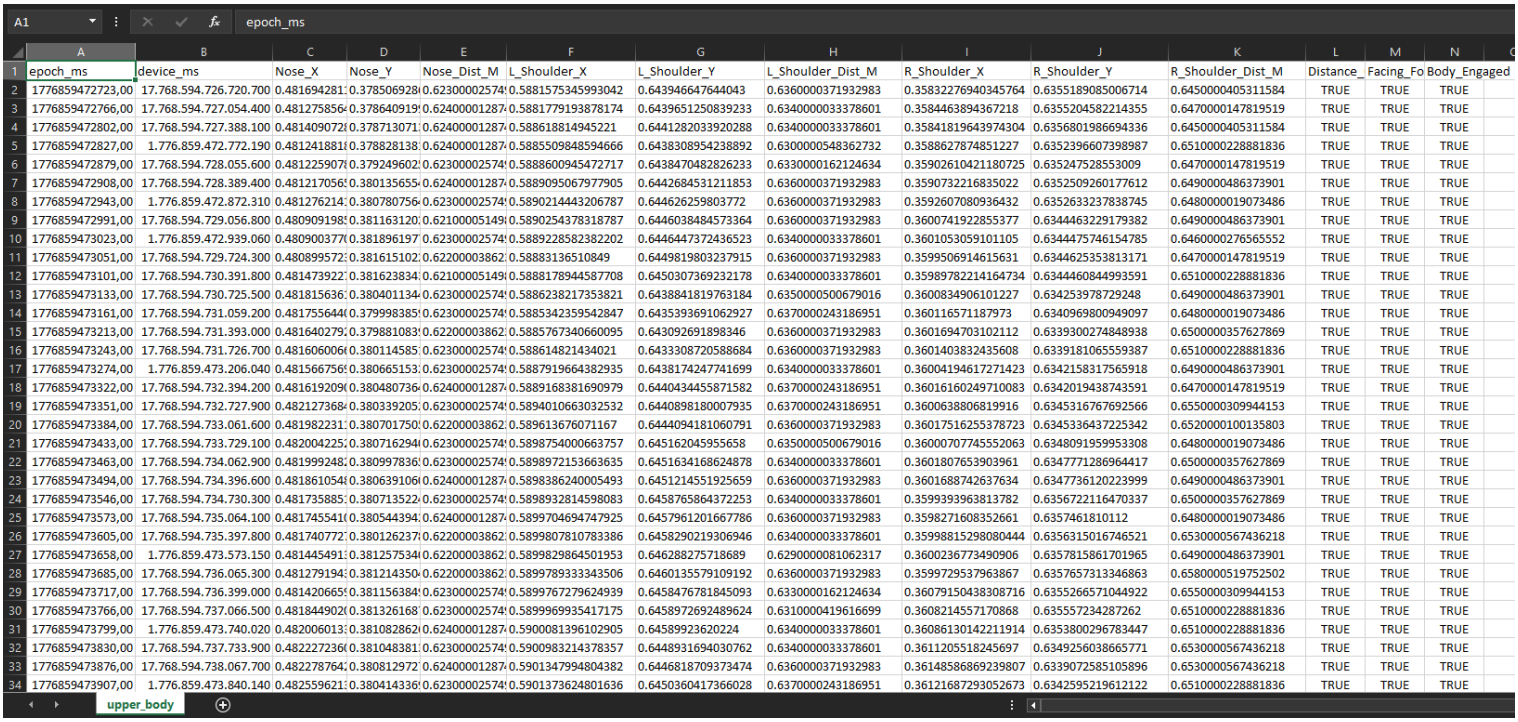
The *tobii_gaze.csv* file records one row per gaze sample at approximately 30 FPS. Each row contains the `epoch_ms` timestamp, the raw screen coordinates `x` and `y` in pixels, and the `on_screen` Boolean classification. Figure 16 shows the first rows of this file from the evaluation session.



	A	B	C	D
1	epoch_ms	x	y	on_screen
2	1776859472746,00	542.161.300.884.796	201.518.989.355.194	TRUE
3	1776859472779,00	54.153.438.743.775	201.252.720.529.811	TRUE
4	1776859472812,00	533.429.712.886.823	203.012.575.822.564	TRUE
5	1776859472845,00	531.903.859.146.547	203.182.852.255.862	TRUE
6	1776859472879,00	533.628.625.017.072	202.604.258.156.839	TRUE
7	1776859472908,00	534.712.673.456.255	201.818.203.643.199	TRUE
8	1776859472943,00	536.161.549.486.195	204.595.748.398.414	TRUE
9	1776859472979,00	536.610.391.157.957	203.922.041.368.951	TRUE
10	1776859473012,00	536.522.723.749.813	203.497.298.904.433	TRUE
11	1776859473046,00	541.005.910.087.079	206.450.976.612.113	TRUE
12	1776859473079,00	541.675.990.381.938	204.265.321.909.843	TRUE
13	1776859473112,00	536.752.505.475.959	191.644.025.511.255	TRUE
14	1776859473146,00	537.997.019.883.118	185.062.980.610.281	TRUE
15	1776859473179,00	537.314.548.789.368	182.100.776.497.918	TRUE
16	1776859473213,00	537.189.161.032.891	183.525.776.393.529	TRUE
17	1776859473245,00	541.408.241.771.243	182.451.973.181.888	TRUE
18	1776859473279,00	542.360.048.191.688	180.763.659.776.506	TRUE
19	1776859473312,00	542.040.942.306.045	179.643.897.112.166	TRUE
20	1776859473345,00	541.556.885.643.276	178.569.855.467.418	TRUE
21	1776859473379,00	541.589.080.685.953	177.997.396.575.738	TRUE
22	1776859473412,00	542.911.665.237.209	174.293.228.346.409	TRUE
23	1776859473445,00	545.332.888.567.207	172.131.095.412.383	TRUE
24	1776859473479,00	544.584.260.732.985	171.583.338.383.244	TRUE
25	1776859473512,00	544.913.605.844.107	171.651.352.325.627	TRUE
26	1776859473546,00	543.223.610.652.713	169.937.878.650.467	TRUE
27	1776859473579,00	541.759.755.384.212	167.077.214.225.026	TRUE
28	1776859473612,00	540.010.508.616.382	164.786.831.156.302	TRUE
29	1776859473645,00	539.171.700.397.173	16.516.710.576.689	TRUE
30	1776859473679,00	537.724.116.796.119	169.031.787.340.453	TRUE
31	1776859473712,00	537.486.615.153.122	16.985.205.134.559	TRUE
32	1776859473746,00	544.913.024.188.043	17.395.397.158.225	TRUE
33	1776859473779,00	544.571.790.935.965	177.226.160.081.776	TRUE
34	1776859473812,00	541.405.899.559.738	179.323.137.837.024	TRUE

Figure 16 - The first rows of tobii_gaze.csv from the evaluation session. The epoch_ms column contains Unix timestamps in milliseconds used for synchronization. The x and y columns record gaze coordinates in screen pixels within the 1920 × 1200 screen space, while the on_screen column indicates whether the gaze point lies within the screen boundaries.

The `upper_body.csv` file records one row per body tracking frame at approximately 30 FPS. Each row contains the `epoch_ms` timestamp, device timestamp, nose and shoulder landmark coordinates and depths, and the three Boolean engagement signals. Figure 17 shows the first rows of this file.



epoch_ms	device_ms	Nose_X	Nose_Y	Nose_Dist_M	L_Shoulder_X	L_Shoulder_Y	L_Shoulder_Dist_M	R_Shoulder_X	R_Shoulder_Y	R_Shoulder_Dist_M	Distance_	Facing_Fo	Body_Engaged
1776859472723,00	17.768.594.726.720.700	0.481694281	0.378506928	0.623000025747	0.588157534599304	0.643946647644043	0.6360000371932983	0.358322769403457	0.635518908500671	0.645000040531158	TRUE	TRUE	TRUE
1776859472766,00	17.768.594.727.054.400	0.481275856	0.378640919	0.624000012877	0.588177919387817	0.643965125083923	0.6340000033378601	0.358446389436721	0.635520458221435	0.647000014781951	TRUE	TRUE	TRUE
1776859472802,00	17.768.594.727.388.100	0.481409072	0.378713071	0.624000012877	0.588618814945221	0.64412820339202	0.6340000033378601	0.358418196439743	0.635680198669433	0.645000040531158	TRUE	TRUE	TRUE
1776859472827,00	1.776.859.472.772.190	0.481241881	0.378828138	0.624000012877	0.588550984859466	0.643830895423892	0.6300000548362732	0.358862787485122	0.635239660739898	0.651000022888183	TRUE	TRUE	TRUE
1776859472879,00	17.768.594.728.055.600	0.481225907	0.379249602	0.623000025747	0.588860094547271	0.643847048282623	0.6330000162124634	0.359026104211807	0.635247528553009	0.647000014781951	TRUE	TRUE	TRUE
1776859472908,00	17.768.594.728.389.400	0.481217056	0.380135655	0.624000012877	0.588909506797790	0.64426845312118	0.6360000371932983	0.35907322168350	0.635250926017761	0.649000048637390	TRUE	TRUE	TRUE
1776859472943,00	1.776.859.472.872.310	0.481276214	0.380780756	0.623000025747	0.589021444320678	0.64462625980377	0.6360000371932983	0.35926070809364	0.635263323783874	0.648000001907348	TRUE	TRUE	TRUE
1776859472991,00	17.768.594.729.056.800	0.480909198	0.381163120	0.621000051497	0.589025437831878	0.64460384845736	0.6360000371932983	0.36007419228555	0.634446322917938	0.649000048637390	TRUE	TRUE	TRUE
1776859473023,00	1.776.859.472.939.060	0.480900377	0.381896197	0.623000025747	0.588922858238220	0.64464473724365	0.6340000033378601	0.36010530591011	0.634447574615478	0.646000027656552	TRUE	TRUE	TRUE
1776859473051,00	17.768.594.729.724.300	0.480899572	0.381615102	0.622000038627	0.58883136510849	0.64498190323791	0.6360000371932983	0.35995069146156	0.634462535381317	0.647000014781951	TRUE	TRUE	TRUE
1776859473101,00	17.768.594.730.391.800	0.481473922	0.381623834	0.621000051497	0.588817894458770	0.64503073692321	0.6340000033378601	0.35989782214164	0.634446084499359	0.651000022888183	TRUE	TRUE	TRUE
1776859473133,00	17.768.594.730.725.500	0.481815636	0.380401134	0.623000025747	0.588623821735382	0.64388418197631	0.6350000500679016	0.36008349061012	0.634253978729248	0.649000048637390	TRUE	TRUE	TRUE
1776859473161,00	17.768.594.731.059.200	0.481755644	0.379998385	0.623000025747	0.588534235954284	0.64353936910629	0.6370000243186951	0.360116571187973	0.634096980094909	0.648000001907348	TRUE	TRUE	TRUE
1776859473213,00	17.768.594.731.393.000	0.481640279	0.379881083	0.622000038627	0.588576734066009	0.64309269189834	0.6360000371932983	0.360169470310211	0.633930027484893	0.650000035762786	TRUE	TRUE	TRUE
1776859473243,00	17.768.594.731.726.700	0.481606006	0.380114585	0.623000025747	0.588614821434021	0.64333087205886	0.6360000371932983	0.360140383243560	0.633918106555938	0.651000022888183	TRUE	TRUE	TRUE
1776859473274,00	1.776.859.473.206.040	0.481566756	0.380665153	0.623000025747	0.588791966438293	0.64381742477416	0.6340000033378601	0.360041946172714	0.634215831756591	0.649000048637390	TRUE	TRUE	TRUE
1776859473322,00	17.768.594.732.394.200	0.481619209	0.380480736	0.624000012877	0.588916838169097	0.64404344557581	0.6370000243186951	0.360161602497100	0.634201943874359	0.647000014781951	TRUE	TRUE	TRUE
1776859473351,00	17.768.594.732.727.900	0.482127368	0.380339205	0.623000025747	0.589401066303253	0.64408981800079	0.6370000243186951	0.360063880681991	0.634531676769256	0.655000030994153	TRUE	TRUE	TRUE
1776859473384,00	17.768.594.733.061.600	0.481982231	0.380701750	0.622000038627	0.589613676071167	0.64440941810607	0.6360000371932983	0.360175162553787	0.634533643722534	0.652000010013580	TRUE	TRUE	TRUE
1776859473433,00	17.768.594.733.729.100	0.482004225	0.380716294	0.623000025747	0.589875400066375	0.64516204595565	0.6350000500679016	0.360070774555206	0.634809195995330	0.648000001907348	TRUE	TRUE	TRUE
1776859473463,00	17.768.594.734.062.900	0.481999248	0.380997836	0.623000025747	0.589897215366363	0.64516341686248	0.6340000033378601	0.360180765390396	0.634777128696441	0.650000035762786	TRUE	TRUE	TRUE
1776859473494,00	17.768.594.734.396.600	0.481861054	0.380639106	0.624000012877	0.589838624000549	0.64512145519256	0.6360000371932983	0.360168874263763	0.634773612022399	0.649000048637390	TRUE	TRUE	TRUE
1776859473546,00	17.768.594.734.730.300	0.481735885	0.380713522	0.623000025747	0.589893281459808	0.64587658643722	0.6340000033378601	0.359939396381378	0.635672211647033	0.650000035762786	TRUE	TRUE	TRUE
1776859473573,00	17.768.594.735.064.100	0.481745541	0.380544394	0.624000012877	0.589970469474792	0.64579612016677	0.6360000371932983	0.35982716083526	0.63574618101	0.648000001907348	TRUE	TRUE	TRUE
1776859473605,00	17.768.594.735.397.800	0.481740772	0.380126237	0.622000038627	0.589980781078338	0.64582902193069	0.6340000033378601	0.359988152980804	0.635631501674652	0.653000056743621	TRUE	TRUE	TRUE
1776859473658,00	1.776.859.473.573.150	0.481445491	0.381257534	0.622000038627	0.589982986450195	0.64628827571868	0.6290000081062317	0.36002367734909	0.635781586170196	0.649000048637390	TRUE	TRUE	TRUE
1776859473685,00	17.768.594.736.065.300	0.481279194	0.381214350	0.622000038627	0.589978933343506	0.64601355791091	0.6360000371932983	0.35997295379638	0.635765731334686	0.658000051975250	TRUE	TRUE	TRUE
1776859473717,00	17.768.594.736.399.000	0.481420665	0.381156384	0.623000025747	0.589976727962493	0.64584767818450	0.6330000162124634	0.360791504383087	0.635526657104492	0.655000030994153	TRUE	TRUE	TRUE
1776859473766,00	17.768.594.737.066.500	0.481844902	0.381326168	0.623000025747	0.589996935417175	0.64589726924896	0.6310000419616699	0.36082145577108	0.63555723428726	0.651000022888183	TRUE	TRUE	TRUE
1776859473799,00	1.776.859.473.740.020	0.482006013	0.381082862	0.624000012877	0.59000813961029	0.64589923620224	0.6340000033378601	0.36086130142211	0.635380029678344	0.651000022888183	TRUE	TRUE	TRUE
1776859473830,00	17.768.594.737.733.900	0.482227236	0.381048381	0.623000025747	0.590098321437835	0.64489316940307	0.6340000033378601	0.36112055182456	0.634925603866577	0.653000056743621	TRUE	TRUE	TRUE
1776859473876,00	17.768.594.738.067.700	0.482278764	0.380812972	0.624000012877	0.590134799480438	0.64468187093734	0.6360000371932983	0.36148586869239	0.633907258510589	0.653000056743621	TRUE	TRUE	TRUE
1776859473907,00	1.776.859.473.840.140	0.482559621	0.380413361	0.623000025747	0.590137362480163	0.64503604173660	0.6370000243186951	0.36121687293052	0.634259521961212	0.651000022888183	TRUE	TRUE	TRUE

Figure 17 - The first rows of upper_body.csv from the evaluation session. The columns Distance_OK, Facing_Forward, and Body_Engaged represent the per frame engagement signals used by the analysis pipeline. Both files share the epoch_ms timestamp reference, enabling nearest neighbour temporal alignment during post session analysis.

5.5. Real Time Body Tracking and Engagement Classification

The body tracking pipeline classifies the student's engagement state on every frame based on two independent signals: distance from the sensor and body orientation. The combination of these signals produces four distinct states, which are displayed in real time in the RealSense preview window and recorded as Boolean fields in `upper_body.csv`. Figures 18 to 21 illustrate each state as captured during the evaluation session.

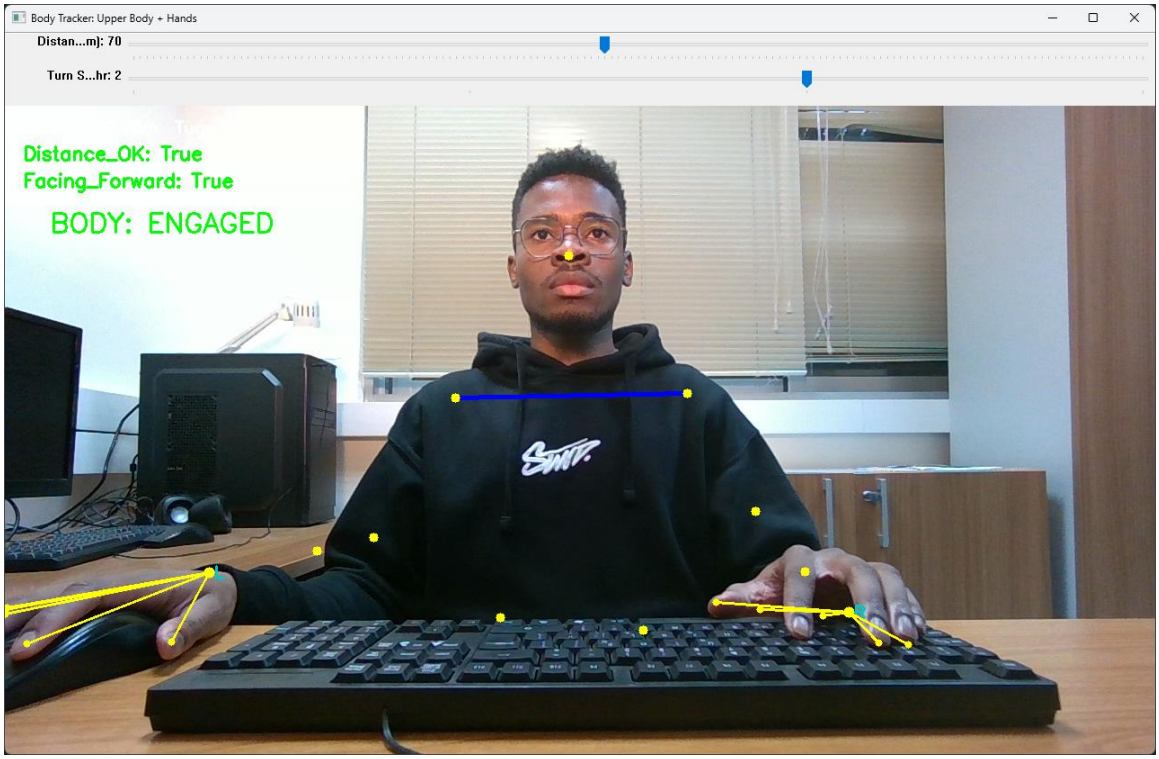


Figure 18 - The body tracking preview window showing the BODY: ENGAGED state. Both Distance_OK and Facing_Forward are True, the shoulder line is approximately horizontal, and the student is within the configured 0.70 m distance threshold.

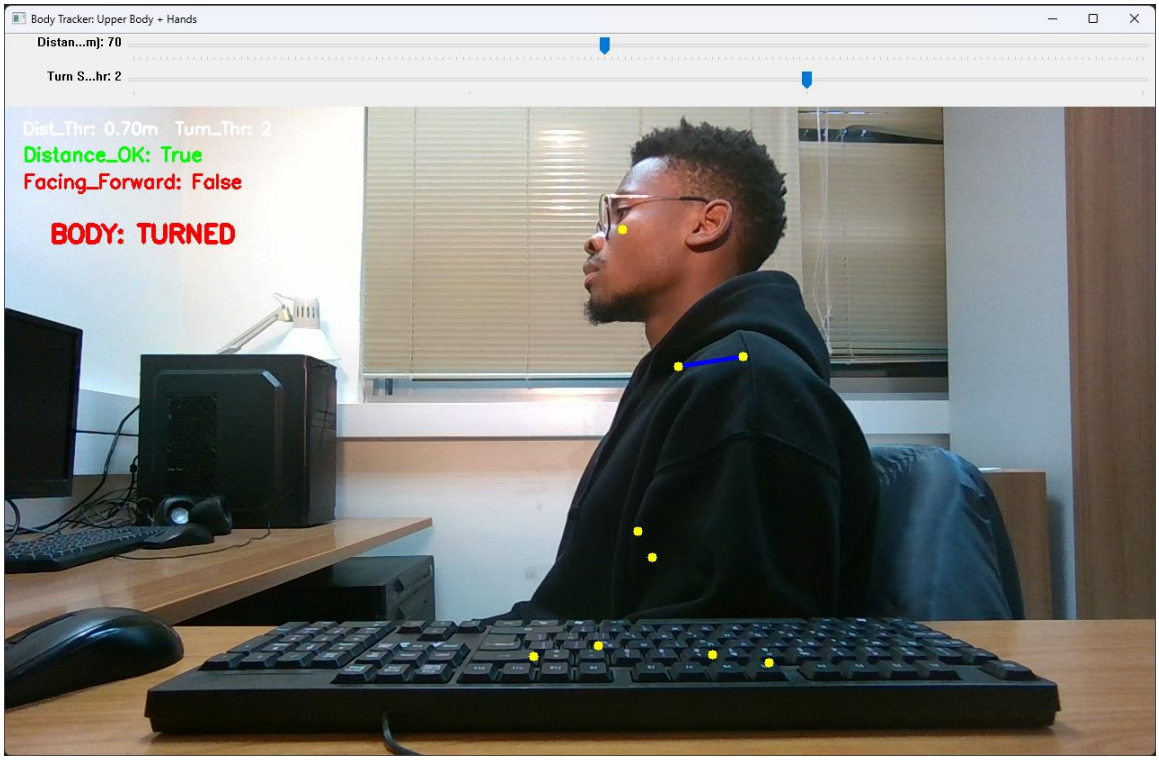


Figure 19 - The BODY: TURNED state. The student has rotated away from the screen while remaining within the acceptable distance range. The shoulder line is visibly angled, and the three cue rotation score exceeds the turn threshold of 2.

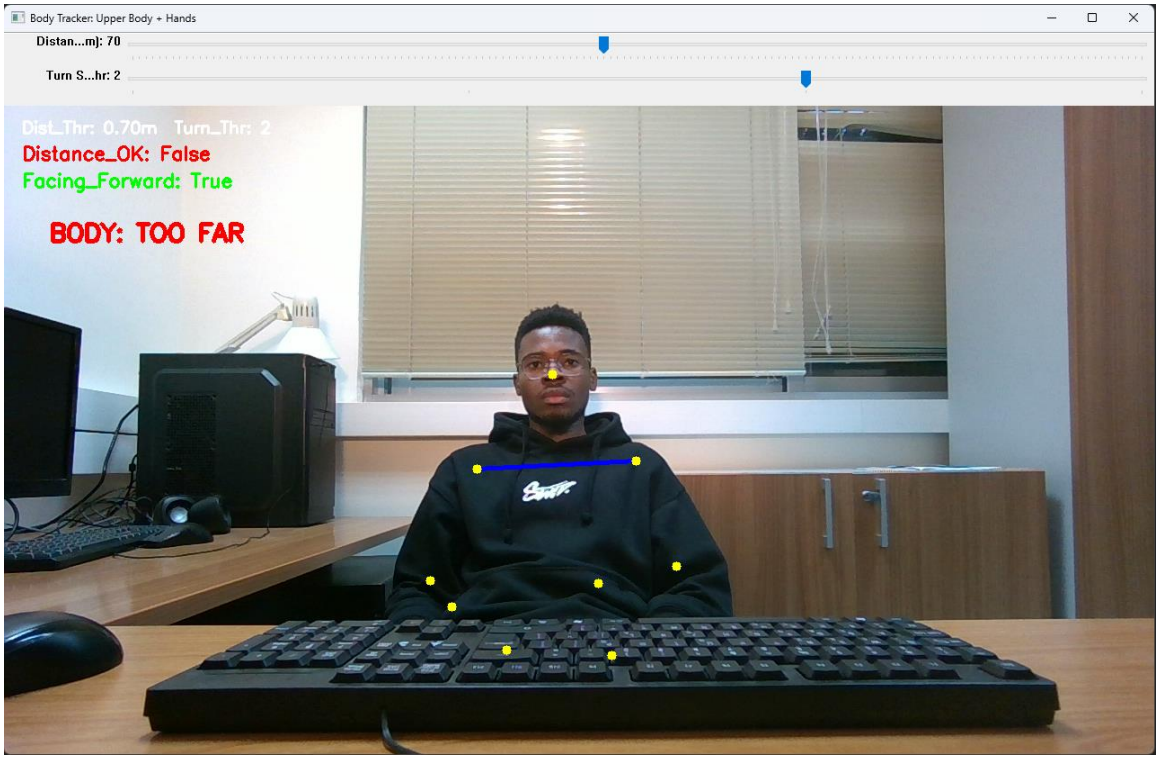


Figure 20 - The BODY: TOO FAR state. The student remains oriented towards the screen but has moved beyond the 0.70 m distance threshold. The shoulder landmarks remain approximately level, correctly distinguishing this condition from a rotation event.

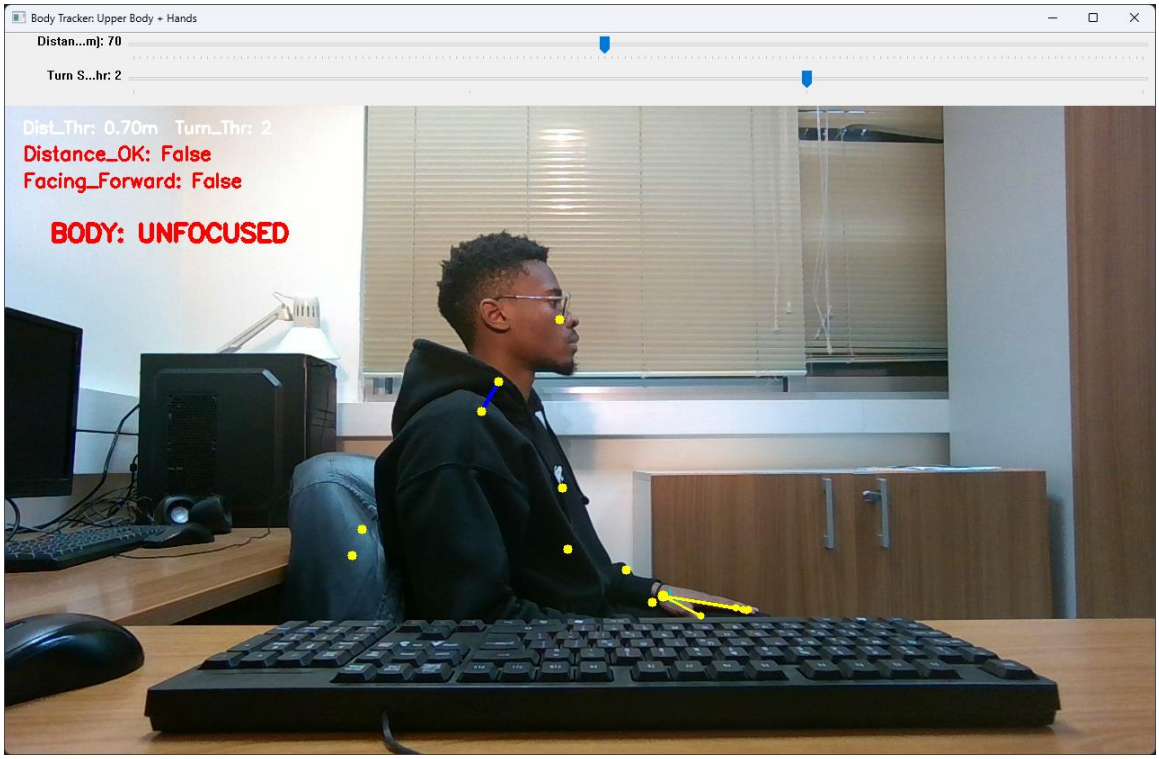


Figure 21 - The BODY: UNFOCUSED state. Both Distance_OK and Facing_Forward are False simultaneously, indicating that the student is both too far from the workstation and not oriented towards the screen.

These four states are classified in real time on every frame and recorded in the `upper_body.csv` output file as the *Distance_OK*, *Facing_Forward*, and *Body_Engaged* Boolean fields.

In addition to upper body pose estimation, the system simultaneously runs MediaPipe Hands on every frame to detect and track hand landmarks. Hand detection provides supplementary behavioral context, capturing fine grained hand activity such as typing, pointing, or reaching toward the robot workspace.

Both hands are detected independently, with finger joint skeletons drawn in yellow and wrist anchors labelled L and R for left and right respectively. Hand presence and landmark coordinates are recorded to *hands_data.csv* at 30 FPS alongside the body tracking stream.

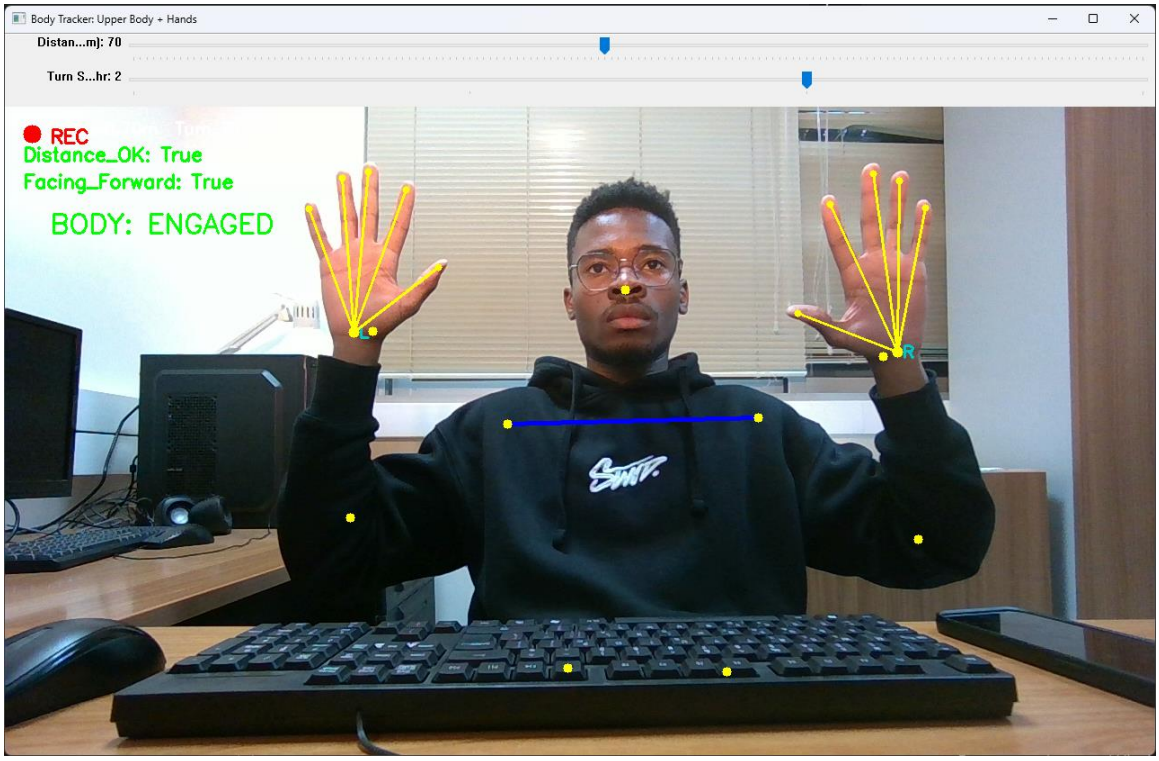


Figure 22 - Hand landmark detection during active recording. The system detects both hands simultaneously using MediaPipe Hands, drawing finger joint skeletons in yellow with wrist anchors labelled L and R. Hand presence data is recorded to hands_data.csv at 30 FPS alongside the body tracking stream.

5.6. Post Session Engagement Analysis

Following the recording, the full analysis pipeline was executed from the Data Analysis tab. The pipeline loaded both CSV files, performed nearest neighbour temporal alignment using `pandas.merge_asof` with a 50 ms tolerance, computed the per frame Focus Index, and generated all analysis visualizations. Figure 23 shows the analysis results panel with the complete session metrics.

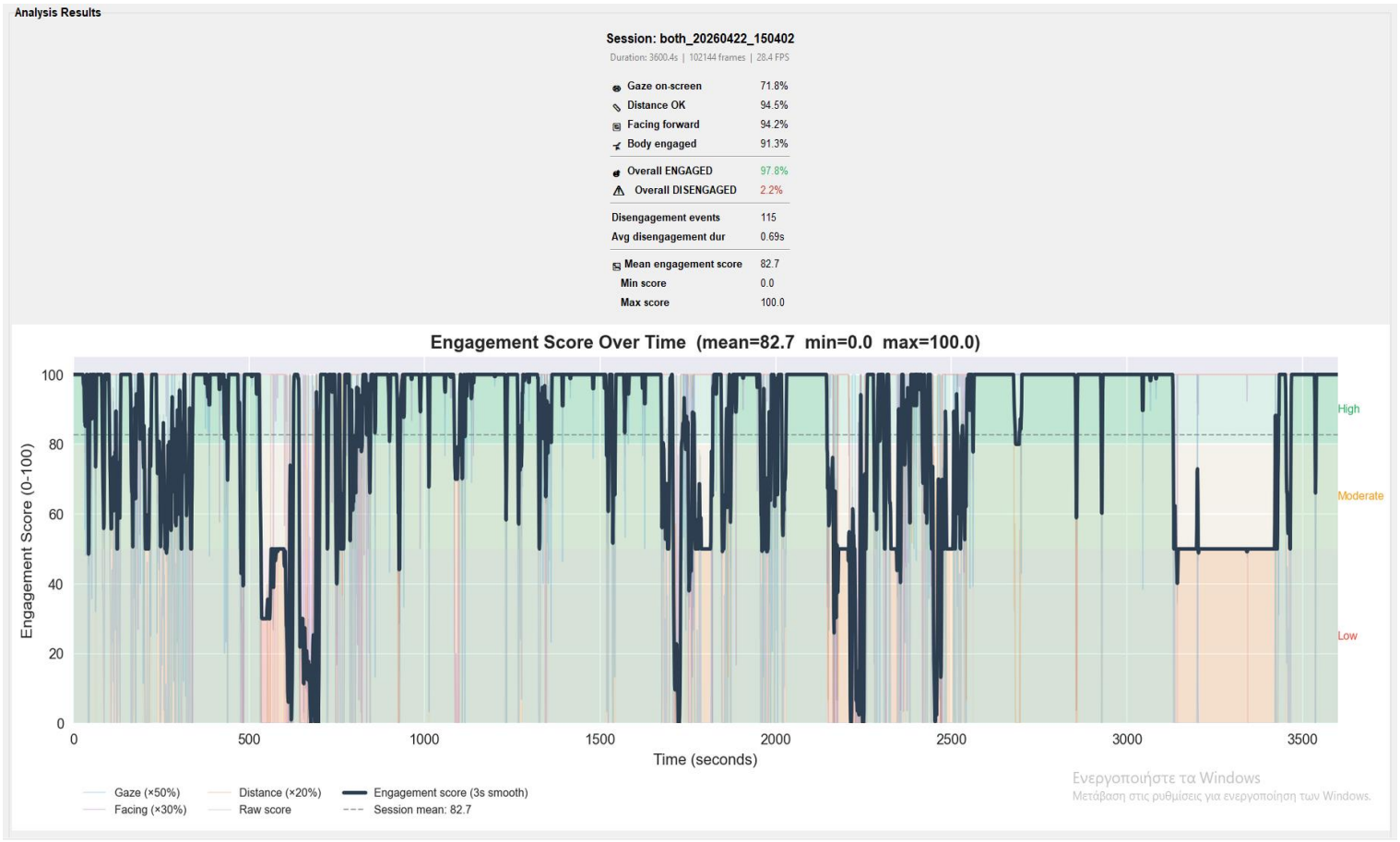


Figure 23 - The analysis results panel showing the complete engagement metrics for the one hour evaluation session. The session duration is 3600.4 seconds, with 102,144 body tracking frames processed at an effective rate of 28.4 FPS.

The key session metrics are summarized in Table 13.

Table 13 - Complete engagement metrics for the one hour evaluation session, including timing, signal distributions, disengagement statistics, and Focus Index summary values.

Metric	Value
Session duration	3600.4 seconds (60.0 minutes)
Total frames analysed	102,144
Effective frame rate	28.4 FPS
Gaze samples collected	108,013 at 30.0 FPS
Sensor start offset	23 milliseconds
Gaze on screen	71.8%
Distance OK	94.5%
Facing forward	94.2%
Body engaged	91.3%
Overall ENGAGED	97.8%
Overall DISENGAGED	2.2%
Disengagement events	115
— Brief (< 2 seconds)	104
— Moderate (2–5 seconds)	8
— Extended (> 5 seconds)	3
Longest disengagement event	6.92 seconds at t = 2245 s
Average disengagement duration	0.69 seconds
Mean engagement score	82.7 / 100
Minimum engagement score	0.0 / 100
Maximum engagement score	100.0 / 100
Time in High zone (≥ 80)	66.8%
Time in Moderate zone (50–79)	26.6%
Time in Low zone (< 50)	6.7%

Figure 24 shows the multimodal engagement timeline, presenting each behavioral signal and the overall engagement verdict as colour coded bands across the full 3600 second session duration.

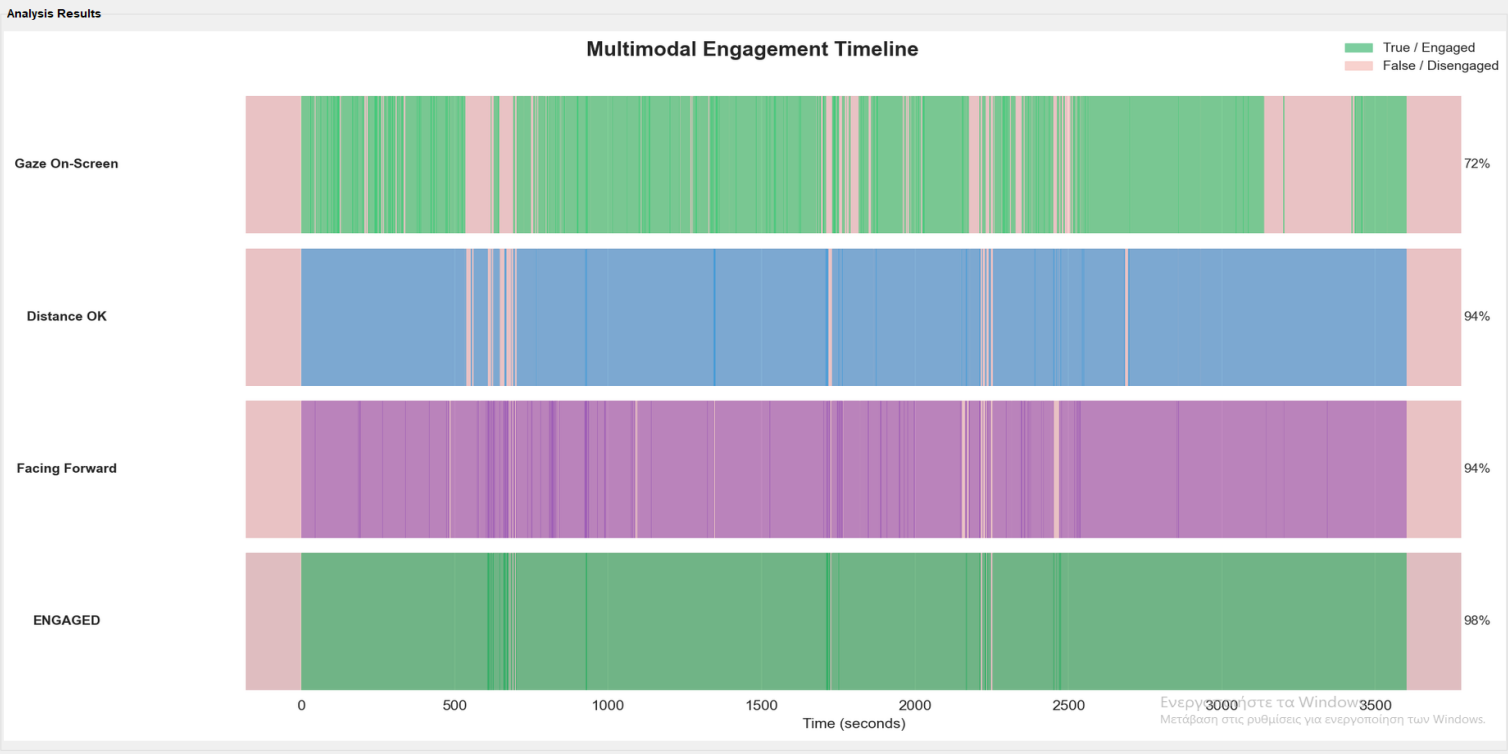


Figure 24 - The multimodal engagement timeline for the one hour evaluation session. Each row represents one behavioral signal: Gaze On Screen, Distance OK, Facing Forward, and the overall ENGAGED verdict. Colored regions indicate active states, while pink regions indicate inactive states. The gaze signal shows the greatest variation, reflecting natural look away behaviour, whereas the distance and orientation signals remain consistently stable.

Figure 25 shows the spatial gaze density heatmap computed from all on screen gaze samples collected during the session.

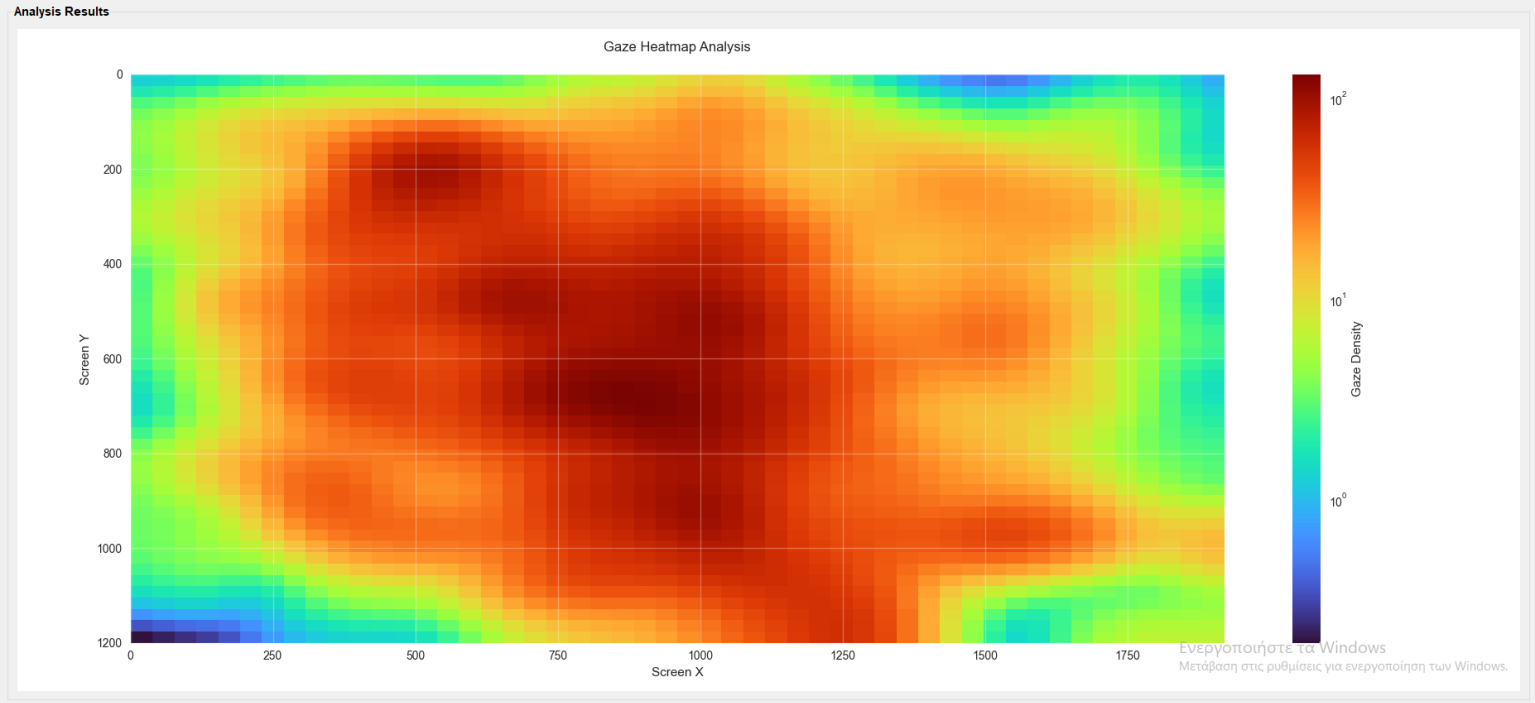


Figure 25 - Spatial gaze density heatmap for the evaluation session. The colour scale ranges from low density (blue) to high density (red). The primary fixation region is concentrated in the upper centre to centre left area of the screen, consistent with the layout of a code development environment where the editor occupies the upper portion of the display.

Figure 26 presents the signal summary bar chart, providing a compact overview of the percentage of session time each behavioral condition was active.

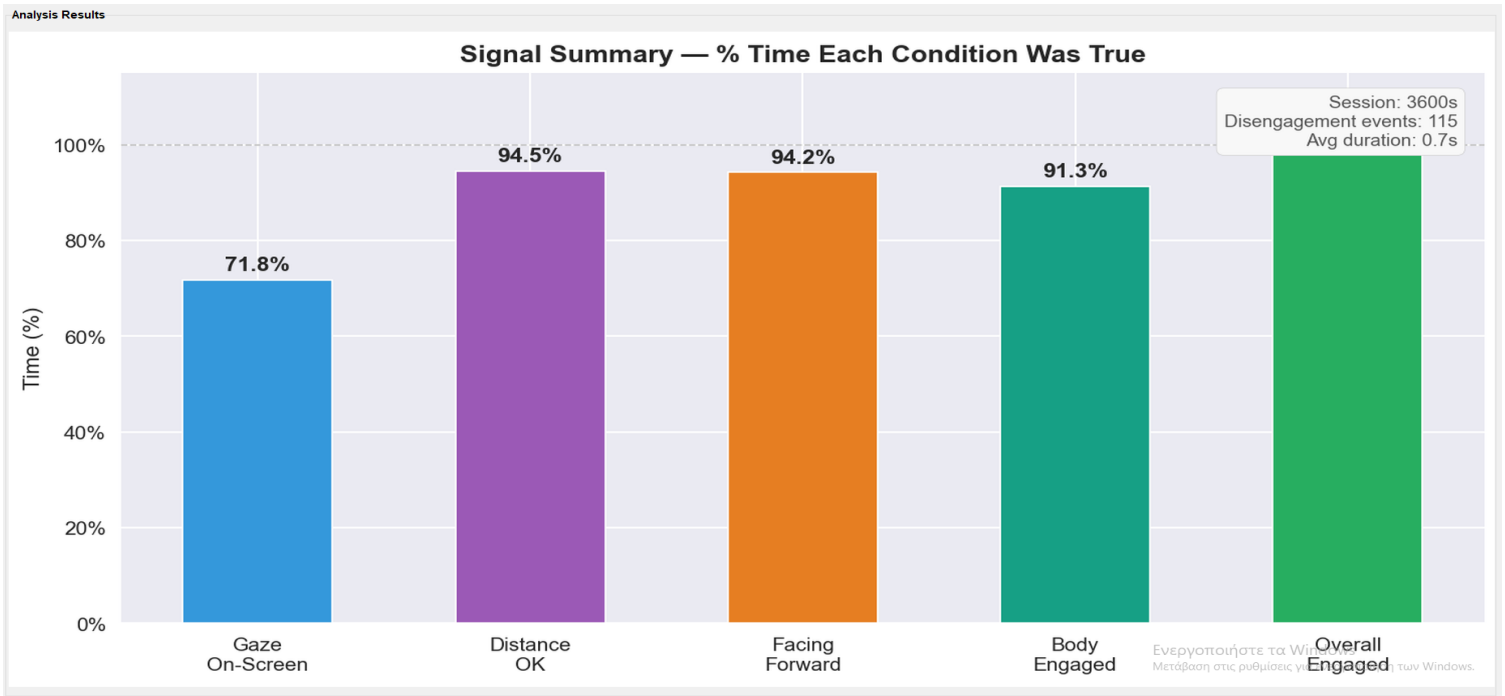


Figure 26 - Signal summary bar chart showing the proportion of session time each behavioral condition was True. Gaze On Screen at 71.8% reflects natural task behaviour including periodic look aways. Distance OK at 94.5% and Facing Forward at 94.2% indicate stable workstation posture. Overall Engaged at 97.8% confirms that the conservative three signal rule suppresses false positives caused by isolated signal variation.

Figure 27 shows the smoothed Focus Index computed across the full session duration.

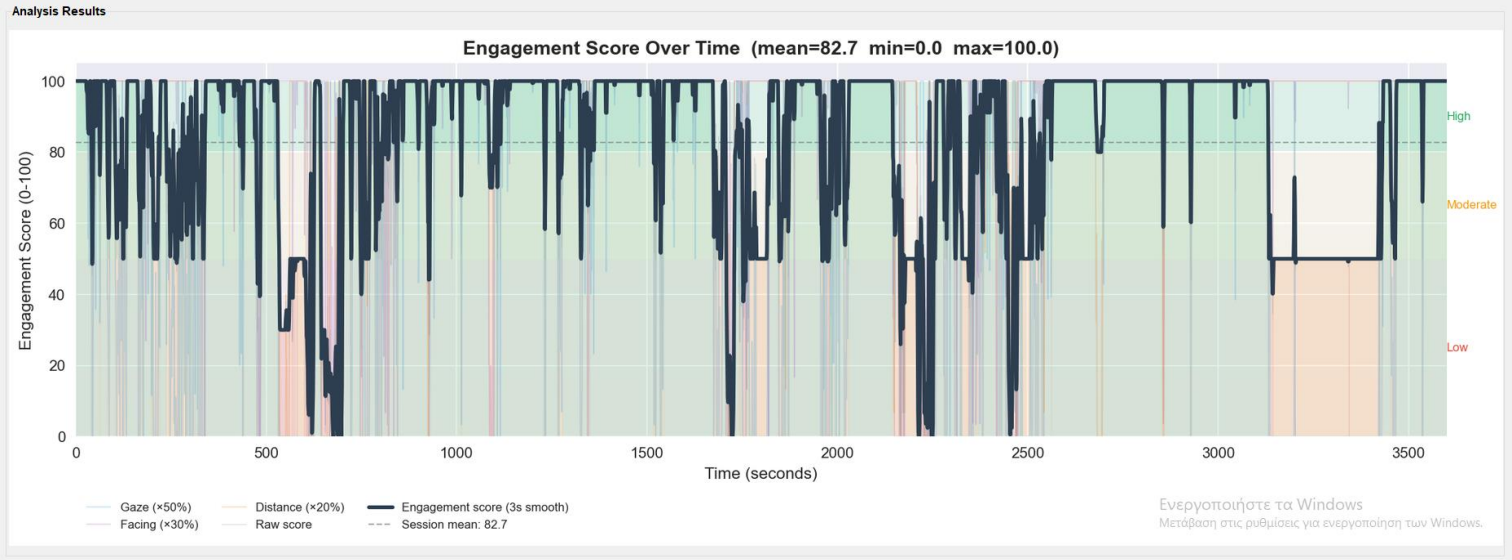
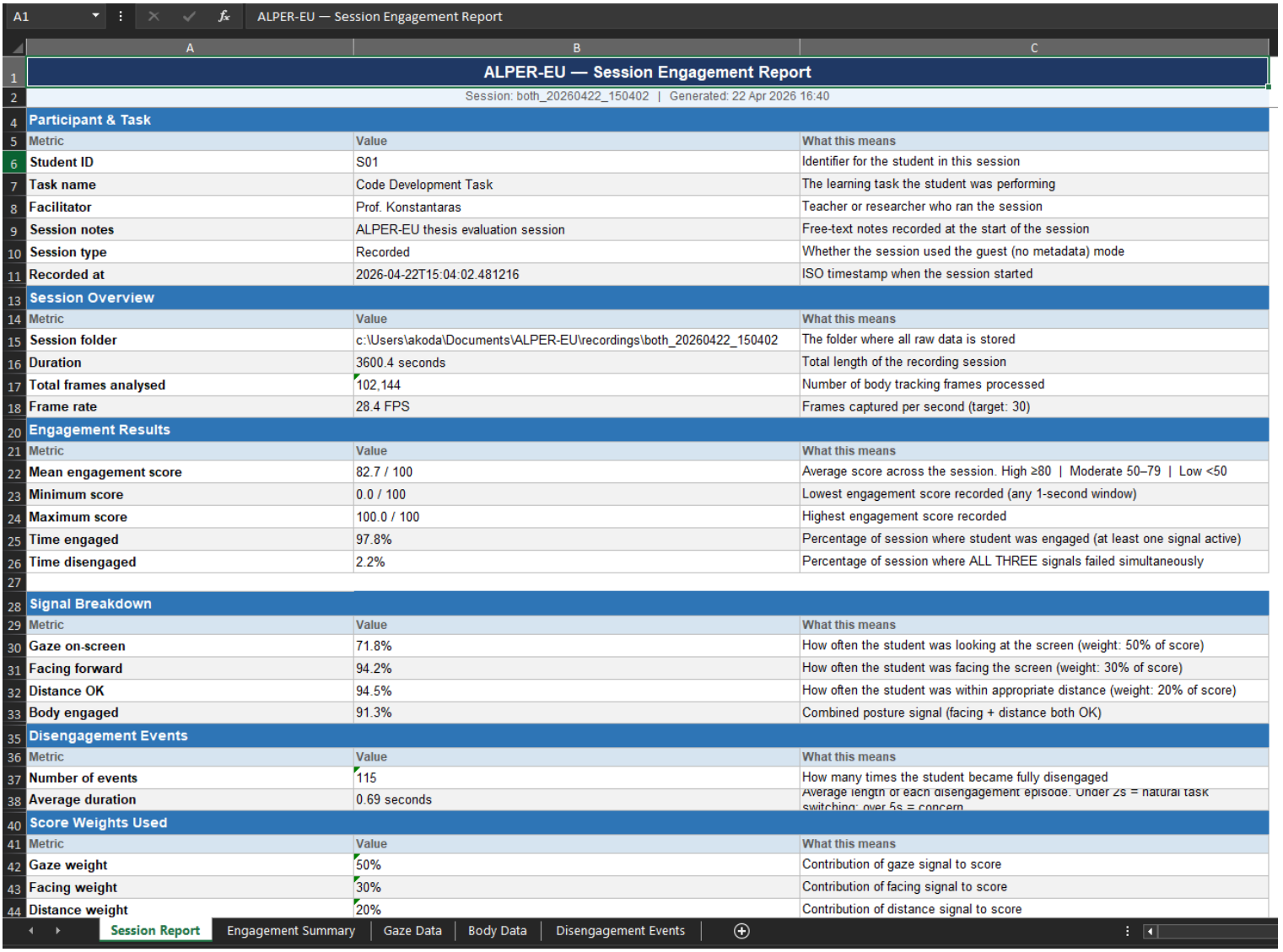


Figure 27 - The smoothed Focus Index across the one hour session. The score is computed per second as a weighted combination of gaze (50%), facing (30%), and distance (20%), and smoothed using a 3 second centred rolling average. The session mean of 82.7 falls within the High engagement zone. Short score drops correspond to brief gaze or posture changes, while the interval between approximately 3000 and 3600 seconds shows sustained moderate engagement, reflecting deliberate postural adjustments towards the end of the session.

5.7. Excel Teacher Report

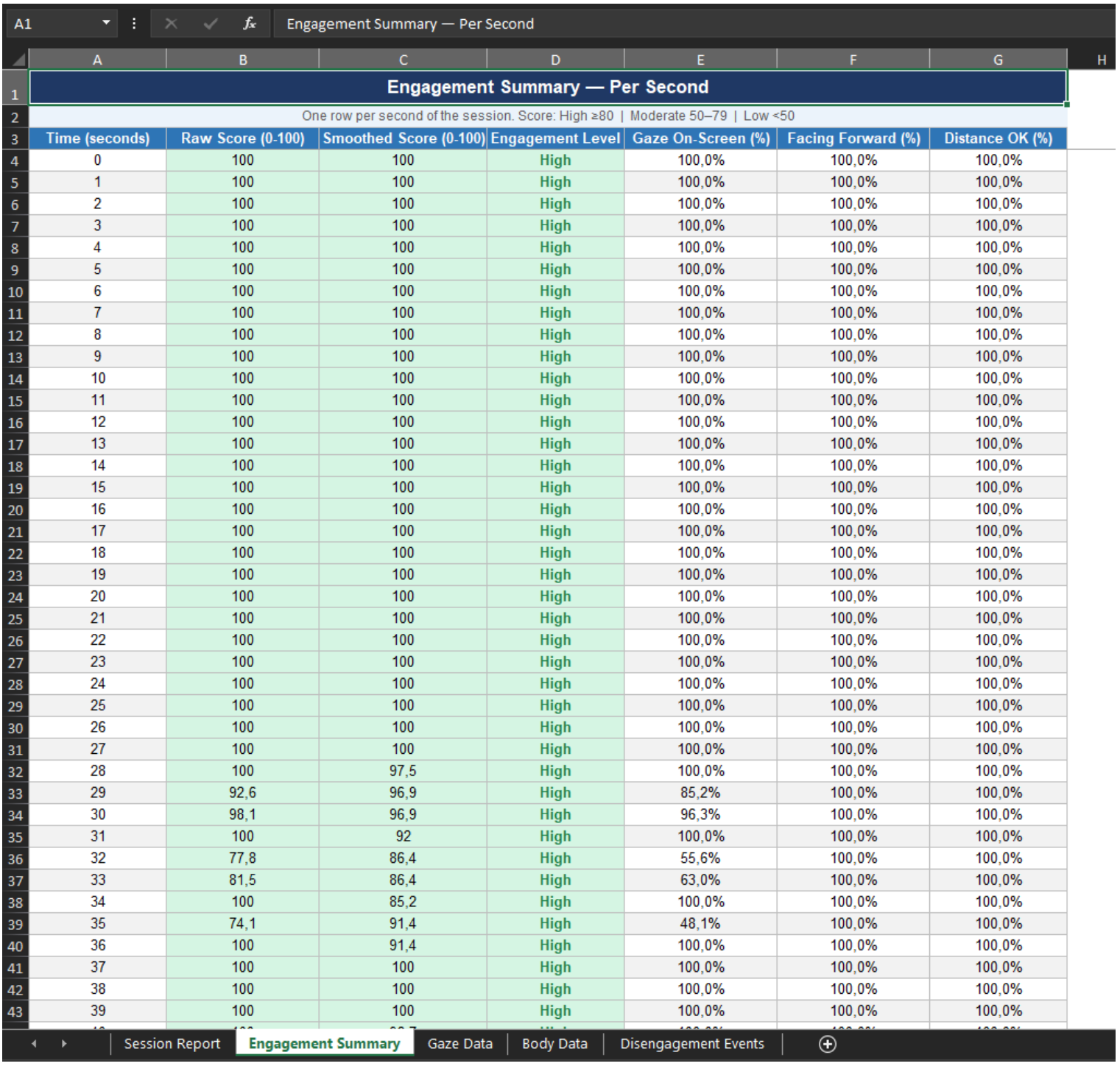
The system generates a structured Excel workbook for teacher review, consisting of five sheets. Figure 28 shows the Session Report sheet, which presents a complete engagement summary in a clear three column format.



ALPER-EU — Session Engagement Report		
Session: both_20260422_150402 | Generated: 22 Apr 2026 16:40		
Participant & Task		
Metric	Value	What this means
Student ID	S01	Identifier for the student in this session
Task name	Code Development Task	The learning task the student was performing
Facilitator	Prof. Konstantaras	Teacher or researcher who ran the session
Session notes	ALPER-EU thesis evaluation session	Free-text notes recorded at the start of the session
Session type	Recorded	Whether the session used the guest (no metadata) mode
Recorded at	2026-04-22T15:04:02.481216	ISO timestamp when the session started
Session Overview		
Metric	Value	What this means
Session folder	c:\Users\akoda\Documents\ALPER-EU\recordings\both_20260422_150402	The folder where all raw data is stored
Duration	3600.4 seconds	Total length of the recording session
Total frames analysed	102,144	Number of body tracking frames processed
Frame rate	28.4 FPS	Frames captured per second (target: 30)
Engagement Results		
Metric	Value	What this means
Mean engagement score	82.7 / 100	Average score across the session. High ≥80 | Moderate 50–79 | Low <50
Minimum score	0.0 / 100	Lowest engagement score recorded (any 1-second window)
Maximum score	100.0 / 100	Highest engagement score recorded
Time engaged	97.8%	Percentage of session where student was engaged (at least one signal active)
Time disengaged	2.2%	Percentage of session where ALL THREE signals failed simultaneously
Signal Breakdown		
Metric	Value	What this means
Gaze on-screen	71.8%	How often the student was looking at the screen (weight: 50% of score)
Facing forward	94.2%	How often the student was facing the screen (weight: 30% of score)
Distance OK	94.5%	How often the student was within appropriate distance (weight: 20% of score)
Body engaged	91.3%	Combined posture signal (facing + distance both OK)
Disengagement Events		
Metric	Value	What this means
Number of events	115	How many times the student became fully disengaged
Average duration	0.69 seconds	Average length of each disengagement episode. Under 2s = natural task switching; over 5s = concern
Score Weights Used		
Metric	Value	What this means
Gaze weight	50%	Contribution of gaze signal to score
Facing weight	30%	Contribution of facing signal to score
Distance weight	20%	Contribution of distance signal to score

Figure 28 - The Session Report sheet of the ALPER-EU Excel report. The report includes session metadata, engagement results, signal breakdown, disengagement statistics, and Focus Index weights, presented using metric name, value, and interpretation columns.

Figure 29 shows the Engagement Summary sheet, which presents the per second Focus Index together with engagement level classification.



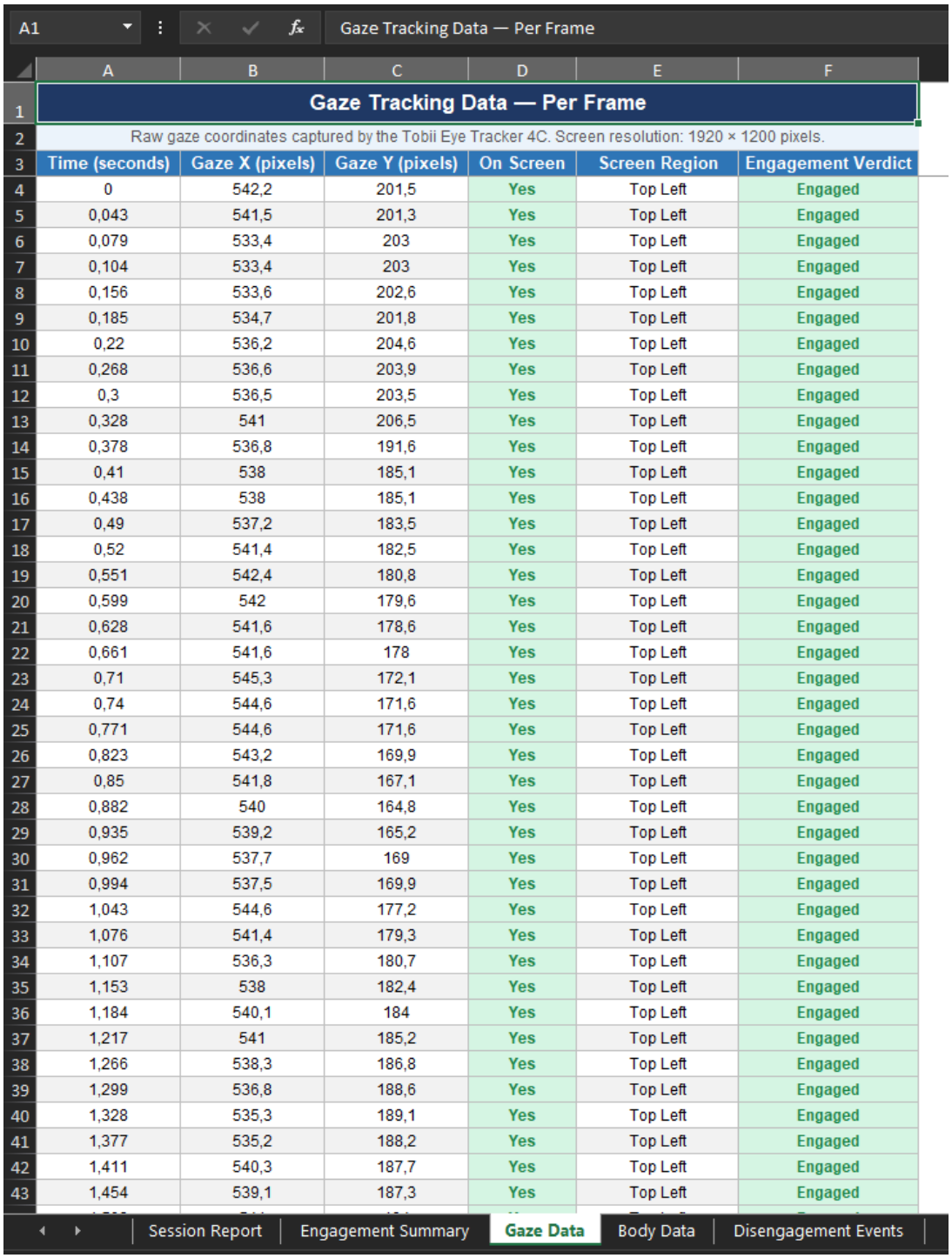
A1 | Engagement Summary — Per Second

	A	B	C	D	E	F	G	H
1	Engagement Summary — Per Second							
2	One row per second of the session. Score: High ≥80 | Moderate 50–79 | Low <50							
3	Time (seconds)	Raw Score (0-100)	Smoothed Score (0-100)	Engagement Level	Gaze On-Screen (%)	Facing Forward (%)	Distance OK (%)	
4	0	100	100	High	100,0%	100,0%	100,0%	
5	1	100	100	High	100,0%	100,0%	100,0%	
6	2	100	100	High	100,0%	100,0%	100,0%	
7	3	100	100	High	100,0%	100,0%	100,0%	
8	4	100	100	High	100,0%	100,0%	100,0%	
9	5	100	100	High	100,0%	100,0%	100,0%	
10	6	100	100	High	100,0%	100,0%	100,0%	
11	7	100	100	High	100,0%	100,0%	100,0%	
12	8	100	100	High	100,0%	100,0%	100,0%	
13	9	100	100	High	100,0%	100,0%	100,0%	
14	10	100	100	High	100,0%	100,0%	100,0%	
15	11	100	100	High	100,0%	100,0%	100,0%	
16	12	100	100	High	100,0%	100,0%	100,0%	
17	13	100	100	High	100,0%	100,0%	100,0%	
18	14	100	100	High	100,0%	100,0%	100,0%	
19	15	100	100	High	100,0%	100,0%	100,0%	
20	16	100	100	High	100,0%	100,0%	100,0%	
21	17	100	100	High	100,0%	100,0%	100,0%	
22	18	100	100	High	100,0%	100,0%	100,0%	
23	19	100	100	High	100,0%	100,0%	100,0%	
24	20	100	100	High	100,0%	100,0%	100,0%	
25	21	100	100	High	100,0%	100,0%	100,0%	
26	22	100	100	High	100,0%	100,0%	100,0%	
27	23	100	100	High	100,0%	100,0%	100,0%	
28	24	100	100	High	100,0%	100,0%	100,0%	
29	25	100	100	High	100,0%	100,0%	100,0%	
30	26	100	100	High	100,0%	100,0%	100,0%	
31	27	100	100	High	100,0%	100,0%	100,0%	
32	28	100	97,5	High	100,0%	100,0%	100,0%	
33	29	92,6	96,9	High	85,2%	100,0%	100,0%	
34	30	98,1	96,9	High	96,3%	100,0%	100,0%	
35	31	100	92	High	100,0%	100,0%	100,0%	
36	32	77,8	86,4	High	55,6%	100,0%	100,0%	
37	33	81,5	86,4	High	63,0%	100,0%	100,0%	
38	34	100	85,2	High	100,0%	100,0%	100,0%	
39	35	74,1	91,4	High	48,1%	100,0%	100,0%	
40	36	100	91,4	High	100,0%	100,0%	100,0%	
41	37	100	100	High	100,0%	100,0%	100,0%	
42	38	100	100	High	100,0%	100,0%	100,0%	
43	39	100	100	High	100,0%	100,0%	100,0%	

Session Report | Engagement Summary | Gaze Data | Body Data | Disengagement Events

Figure 29 - The Engagement Summary sheet showing per second raw score, smoothed score, engagement level, and per signal percentage values. The initial seconds show maximum scores, corresponding to stable engagement immediately after recording begins.

Figure 30 shows the Gaze Data sheet, which presents per frame gaze tracking records.



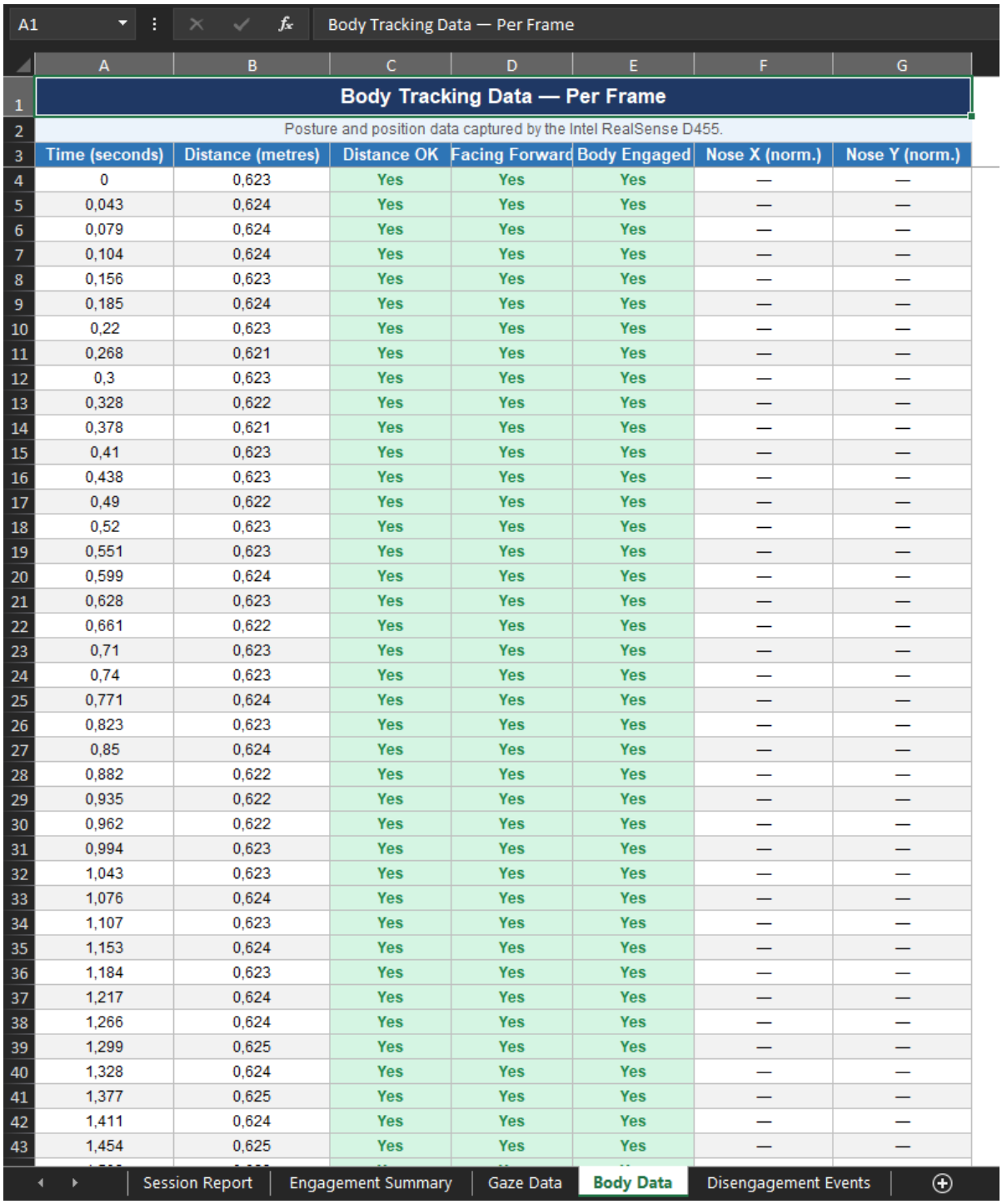
A1 | Gaze Tracking Data — Per Frame

	A	B	C	D	E	F
1	Gaze Tracking Data — Per Frame					
2	Raw gaze coordinates captured by the Tobii Eye Tracker 4C. Screen resolution: 1920 × 1200 pixels.					
3	Time (seconds)	Gaze X (pixels)	Gaze Y (pixels)	On Screen	Screen Region	Engagement Verdict
4	0	542,2	201,5	Yes	Top Left	Engaged
5	0,043	541,5	201,3	Yes	Top Left	Engaged
6	0,079	533,4	203	Yes	Top Left	Engaged
7	0,104	533,4	203	Yes	Top Left	Engaged
8	0,156	533,6	202,6	Yes	Top Left	Engaged
9	0,185	534,7	201,8	Yes	Top Left	Engaged
10	0,22	536,2	204,6	Yes	Top Left	Engaged
11	0,268	536,6	203,9	Yes	Top Left	Engaged
12	0,3	536,5	203,5	Yes	Top Left	Engaged
13	0,328	541	206,5	Yes	Top Left	Engaged
14	0,378	536,8	191,6	Yes	Top Left	Engaged
15	0,41	538	185,1	Yes	Top Left	Engaged
16	0,438	538	185,1	Yes	Top Left	Engaged
17	0,49	537,2	183,5	Yes	Top Left	Engaged
18	0,52	541,4	182,5	Yes	Top Left	Engaged
19	0,551	542,4	180,8	Yes	Top Left	Engaged
20	0,599	542	179,6	Yes	Top Left	Engaged
21	0,628	541,6	178,6	Yes	Top Left	Engaged
22	0,661	541,6	178	Yes	Top Left	Engaged
23	0,71	545,3	172,1	Yes	Top Left	Engaged
24	0,74	544,6	171,6	Yes	Top Left	Engaged
25	0,771	544,6	171,6	Yes	Top Left	Engaged
26	0,823	543,2	169,9	Yes	Top Left	Engaged
27	0,85	541,8	167,1	Yes	Top Left	Engaged
28	0,882	540	164,8	Yes	Top Left	Engaged
29	0,935	539,2	165,2	Yes	Top Left	Engaged
30	0,962	537,7	169	Yes	Top Left	Engaged
31	0,994	537,5	169,9	Yes	Top Left	Engaged
32	1,043	544,6	177,2	Yes	Top Left	Engaged
33	1,076	541,4	179,3	Yes	Top Left	Engaged
34	1,107	536,3	180,7	Yes	Top Left	Engaged
35	1,153	538	182,4	Yes	Top Left	Engaged
36	1,184	540,1	184	Yes	Top Left	Engaged
37	1,217	541	185,2	Yes	Top Left	Engaged
38	1,266	538,3	186,8	Yes	Top Left	Engaged
39	1,299	536,8	188,6	Yes	Top Left	Engaged
40	1,328	535,3	189,1	Yes	Top Left	Engaged
41	1,377	535,2	188,2	Yes	Top Left	Engaged
42	1,411	540,3	187,7	Yes	Top Left	Engaged
43	1,454	539,1	187,3	Yes	Top Left	Engaged

Session Report | Engagement Summary | Gaze Data | Body Data | Disengagement Events

Figure 30 - The Gaze Data sheet presenting per frame gaze coordinates, including time in seconds, X and Y positions, on screen classification, derived screen region, and engagement verdict.

Figure 31 shows the Body Data sheet, which presents per frame body tracking signals.



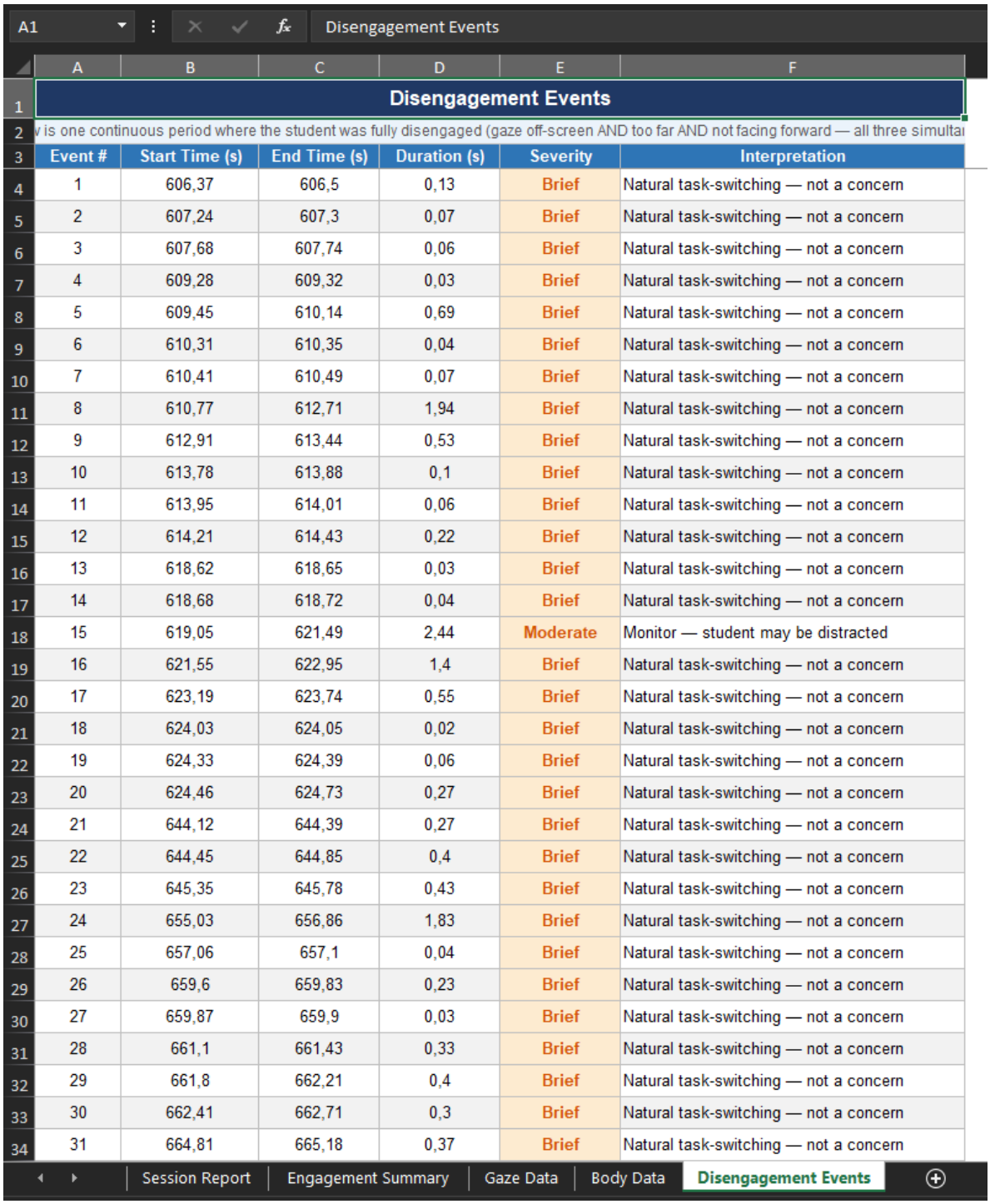
A1 | Body Tracking Data — Per Frame

	A	B	C	D	E	F	G
1	Body Tracking Data — Per Frame						
2	Posture and position data captured by the Intel RealSense D455.						
3	Time (seconds)	Distance (metres)	Distance OK	Facing Forward	Body Engaged	Nose X (norm.)	Nose Y (norm.)
4	0	0,623	Yes	Yes	Yes	—	—
5	0,043	0,624	Yes	Yes	Yes	—	—
6	0,079	0,624	Yes	Yes	Yes	—	—
7	0,104	0,624	Yes	Yes	Yes	—	—
8	0,156	0,623	Yes	Yes	Yes	—	—
9	0,185	0,624	Yes	Yes	Yes	—	—
10	0,22	0,623	Yes	Yes	Yes	—	—
11	0,268	0,621	Yes	Yes	Yes	—	—
12	0,3	0,623	Yes	Yes	Yes	—	—
13	0,328	0,622	Yes	Yes	Yes	—	—
14	0,378	0,621	Yes	Yes	Yes	—	—
15	0,41	0,623	Yes	Yes	Yes	—	—
16	0,438	0,623	Yes	Yes	Yes	—	—
17	0,49	0,622	Yes	Yes	Yes	—	—
18	0,52	0,623	Yes	Yes	Yes	—	—
19	0,551	0,623	Yes	Yes	Yes	—	—
20	0,599	0,624	Yes	Yes	Yes	—	—
21	0,628	0,623	Yes	Yes	Yes	—	—
22	0,661	0,622	Yes	Yes	Yes	—	—
23	0,71	0,623	Yes	Yes	Yes	—	—
24	0,74	0,623	Yes	Yes	Yes	—	—
25	0,771	0,624	Yes	Yes	Yes	—	—
26	0,823	0,623	Yes	Yes	Yes	—	—
27	0,85	0,624	Yes	Yes	Yes	—	—
28	0,882	0,622	Yes	Yes	Yes	—	—
29	0,935	0,622	Yes	Yes	Yes	—	—
30	0,962	0,622	Yes	Yes	Yes	—	—
31	0,994	0,623	Yes	Yes	Yes	—	—
32	1,043	0,623	Yes	Yes	Yes	—	—
33	1,076	0,624	Yes	Yes	Yes	—	—
34	1,107	0,623	Yes	Yes	Yes	—	—
35	1,153	0,624	Yes	Yes	Yes	—	—
36	1,184	0,623	Yes	Yes	Yes	—	—
37	1,217	0,624	Yes	Yes	Yes	—	—
38	1,266	0,624	Yes	Yes	Yes	—	—
39	1,299	0,625	Yes	Yes	Yes	—	—
40	1,328	0,624	Yes	Yes	Yes	—	—
41	1,377	0,625	Yes	Yes	Yes	—	—
42	1,411	0,624	Yes	Yes	Yes	—	—
43	1,454	0,625	Yes	Yes	Yes	—	—

Session Report | Engagement Summary | Gaze Data | Body Data | Disengagement Events

Figure 31 - The Body Data sheet presenting per frame posture and position data, including estimated distance and the three engagement signals: Distance OK, Facing Forward, and Body Engaged.

Figure 32 shows the Disengagement Events sheet.



Disengagement Events

Disengagement Events					
v is one continuous period where the student was fully disengaged (gaze off-screen AND too far AND not facing forward — all three simultar					
Event #	Start Time (s)	End Time (s)	Duration (s)	Severity	Interpretation
1	606,37	606,5	0,13	Brief	Natural task-switching — not a concern
2	607,24	607,3	0,07	Brief	Natural task-switching — not a concern
3	607,68	607,74	0,06	Brief	Natural task-switching — not a concern
4	609,28	609,32	0,03	Brief	Natural task-switching — not a concern
5	609,45	610,14	0,69	Brief	Natural task-switching — not a concern
6	610,31	610,35	0,04	Brief	Natural task-switching — not a concern
7	610,41	610,49	0,07	Brief	Natural task-switching — not a concern
8	610,77	612,71	1,94	Brief	Natural task-switching — not a concern
9	612,91	613,44	0,53	Brief	Natural task-switching — not a concern
10	613,78	613,88	0,1	Brief	Natural task-switching — not a concern
11	613,95	614,01	0,06	Brief	Natural task-switching — not a concern
12	614,21	614,43	0,22	Brief	Natural task-switching — not a concern
13	618,62	618,65	0,03	Brief	Natural task-switching — not a concern
14	618,68	618,72	0,04	Brief	Natural task-switching — not a concern
15	619,05	621,49	2,44	Moderate	Monitor — student may be distracted
16	621,55	622,95	1,4	Brief	Natural task-switching — not a concern
17	623,19	623,74	0,55	Brief	Natural task-switching — not a concern
18	624,03	624,05	0,02	Brief	Natural task-switching — not a concern
19	624,33	624,39	0,06	Brief	Natural task-switching — not a concern
20	624,46	624,73	0,27	Brief	Natural task-switching — not a concern
21	644,12	644,39	0,27	Brief	Natural task-switching — not a concern
22	644,45	644,85	0,4	Brief	Natural task-switching — not a concern
23	645,35	645,78	0,43	Brief	Natural task-switching — not a concern
24	655,03	656,86	1,83	Brief	Natural task-switching — not a concern
25	657,06	657,1	0,04	Brief	Natural task-switching — not a concern
26	659,6	659,83	0,23	Brief	Natural task-switching — not a concern
27	659,87	659,9	0,03	Brief	Natural task-switching — not a concern
28	661,1	661,43	0,33	Brief	Natural task-switching — not a concern
29	661,8	662,21	0,4	Brief	Natural task-switching — not a concern
30	662,41	662,71	0,3	Brief	Natural task-switching — not a concern
31	664,81	665,18	0,37	Brief	Natural task-switching — not a concern

Figure 32 - The Disengagement Events sheet listing all 115 disengagement episodes, including start time, end time, duration, severity classification, and interpretation. The majority of events are classified as brief, reflecting natural task switching, with an average duration of 0.69 seconds.

5.8. System Stability and Performance

The one hour continuous recording session demonstrates that the system operates stably over extended durations without data loss or pipeline failures. The body tracking pipeline maintained an effective rate of 28.4 FPS against a target of 30 FPS throughout the session, representing a 5.3% deviation attributable to normal processing overhead from MediaPipe inference and video encoding.

Of the 102,144 body frames recorded, 3,479 exhibited inter frame intervals exceeding 50 milliseconds, corresponding to 3.4% of all frames, while only 43 frames exceeded 100 milliseconds. No frame gaps exceeded 200 milliseconds more than 23 times across the full hour, confirming that the pipeline remained stable without sustained interruption.

The gaze pipeline maintained 30.0 FPS with a standard deviation of only 1.7 milliseconds per frame interval, demonstrating high timing consistency in the software downsampling mechanism.

The two sensing pipelines started within 23 milliseconds of each other, well within the 50 millisecond merge tolerance used during analysis. This confirms that the gate based synchronization mechanism achieves sub frame alignment in practice without requiring hardware synchronization.

The video output size of 1997.3 MB for a 3600 second session at 30 FPS in XVID format is consistent with expected encoding performance for this configuration. All output files were successfully generated and verified, and the analysis pipeline processed all 102,144 frames without errors.

These results confirm that the system satisfies the core technical requirements defined in the User Needs Analysis, including synchronized multimodal acquisition at 30 FPS, millisecond resolution timestamp alignment, structured CSV output, synchronized video recording, and reliable session operation over classroom length durations.

5.9. Discussion of Results

The evaluation session produced engagement metrics consistent with expected behaviour for a focused individual performing a sustained one hour coding task. The mean Focus Index of 82.7 places the session in the High engagement zone, with 66.8% of session time spent at or above the High threshold of 80, 26.6% in the Moderate zone, and only 6.7% in the Low zone. These proportions reflect predominantly attentive posture and screen directed gaze throughout the session, with natural variation arising from task activity.

The per quarter breakdown of the smoothed score reveals a characteristic pattern. The first quarter, from 0 to 15 minutes, has a mean of 78.2, slightly below the High threshold

as the participant settles into the task. The second quarter, from 15 to 30 minutes, reaches peak engagement with a mean of 90.8 as the task becomes familiar. The third quarter, from 30 to 45 minutes, shows a slight decline to 79.2, consistent with the onset of fatigue. The final quarter, from 45 to 60 minutes, recovers partially to a mean of 82.5. This trajectory aligns with established models of sustained attention, which predict an initial adaptation period, a mid session performance peak, and a gradual decline toward the end.

The gaze on screen rate of 71.8% is notably lower than the body based signals, which both remain above 94%. This difference reflects the fundamental behavioural distinction between gaze and posture during programming activity. Gaze shifts rapidly and continuously, while body orientation changes more slowly. The 361 distinct off screen gaze runs, with a mean duration of 2.81 seconds, confirm that gaze variability is dominated by brief, frequent transitions rather than sustained inattention.

The most analytically significant event occurs at approximately 3202 seconds. During this interval, gaze remains off screen for approximately 202 seconds. Despite this, Distance OK remains at 100.0% and Facing Forward at 99.9%, resulting in an Engaged verdict for all corresponding frames and a mean score of 50.3, within the Moderate zone. This event provides a clear demonstration of the three signal rule. A gaze only system would classify this period as disengaged, whereas the multimodal system correctly identifies the student as engaged based on sustained physical presence and orientation. This behaviour is consistent with real task conditions, such as interacting with a physical robot or referencing materials off screen.

The 115 disengagement events are distributed as 104 Brief, 8 Moderate, and 3 Extended episodes. All but the Extended events correspond to natural task switching behaviour. Notably, all disengagement events occur within the first three quarters of the session, while none occur in the final quarter. This is explained by the three signal rule: although gaze is absent for extended periods in the final quarter, body signals remain active, preventing false disengagement classification.

The three Extended events represent the only instances of full disengagement, with the longest lasting 6.92 seconds at 2245 seconds. Each corresponds to a clear physical withdrawal from the workstation.

These findings support four key conclusions. First, the Focus Index provides a stable and meaningful representation of behavioural engagement over extended sessions. Second, the three signal disengagement rule effectively suppresses false positives arising from natural single signal variation. Third, the 23 millisecond sensor start offset confirms that the synchronization mechanism achieves reliable sub frame alignment. Fourth, the system maintains stable operation over a full one hour session, confirming suitability for classroom length deployment.

5.10. System Limitations and Directions for Improvement

The evaluation session highlights several limitations and opportunities for further development.

The body tracking pipeline exhibits higher timing variability compared to the gaze pipeline due to the computational demands of MediaPipe inference and video encoding. Although only 3.4% of frames exceed 50 milliseconds and the impact on analysis is minimal, this variability could become more significant in longer sessions or on lower performance hardware. A potential improvement is to decouple pose estimation from video encoding using separate processing threads.

The evaluation was conducted with a single participant in a controlled environment. The system has not yet been validated in real classroom conditions, where multiple students, environmental variability, and collaborative interactions may introduce more complex behavioural patterns. Future evaluation with larger and more diverse participant groups is required.

The fixed distance threshold of 0.70 metres may not fully account for natural variation in individual posture. The observed mean distance of 0.582 metres with a standard deviation of 0.108 metres suggests that an adaptive, per session calibration approach could improve classification accuracy.

The gaze on screen rate of 71.8% indicates that a substantial portion of gaze behaviour occurs outside the screen. While the multimodal design correctly handles this, the current system cannot distinguish between productive off screen gaze, such as interaction with a robot, and unproductive distraction. Extending the sensing setup to include a wider field of view or additional cameras could address this limitation.

Finally, the evaluation lacks ground truth annotation for engagement states. Without independently labelled data, it is not possible to compute formal classification metrics such as precision or recall. A future study involving manual annotation or observer based evaluation is necessary to quantitatively validate system performance.

Despite these limitations, the system provides a robust foundation for multimodal engagement analysis and demonstrates clear potential for further refinement and real world deployment.

6. Conclusions & Future Work

6.1. Conclusion

This thesis presented the design and implementation of the ALPER-EU Multimodal Engagement Assessment System, a sensor based platform for objective measurement of student behavioral engagement during educational robotics and programming activities at the Hellenic Mediterranean University. The work addressed a key limitation of existing unimodal monitoring approaches, namely the inability to distinguish task related off screen activity, such as robot interaction, from genuine disengagement, by integrating two complementary sensing modalities and applying a conservative and context aware behavioral inference model (Bosch et al., 2015; Monkaresi et al., 2017).

The system was developed across three structured phases. The User Needs Analysis identified seven core requirements, establishing the pedagogical and operational constraints that guided all design decisions. The Design Phase translated these requirements into a modular layered architecture covering sensing, synchronization, processing, inference, storage, and user interaction, with full traceability between requirements and system components. The Implementation Phase realized this architecture as a deployable Python application organized across four primary modules: `MainApp.py`, `GazeAppAlpha.py`, `body-tracker.py`, and `engagement_analysis.py`.

The resulting system satisfies all functional requirements established during the User Needs Analysis. It provides single button session control with fully automated synchronized recording, objective assessment of screen directed behavioral engagement through a transparent rule based Focus Index, context aware multimodal interpretation that distinguishes robot interaction from disengagement through the three signal rule, millisecond level temporal alignment through a two stage synchronization mechanism, synchronized video for post session validation, and structured and reproducible session data storage in modality specific CSV files.

The Focus Index quantifies screen directed behavioral engagement as a weighted linear combination of gaze direction, body orientation, and student proximity, producing a continuous score interpreted across three zones: High, Moderate, and Low. This formulation is grounded in established multimodal engagement research and was implemented as a rule based model to ensure transparency, interpretability, and ease of validation in the current phase. A preliminary evaluation within an ALPER-EU session indicated that the system produced engagement patterns consistent with teacher observations and correctly avoided false disengagement classifications during robot interaction phases.

The system demonstrates that commercially available low cost hardware, including the Tobii Eye Tracker 4C and the Intel RealSense D455, can be combined with open source Python tools to produce a functional, interpretable, and classroom deployable

engagement assessment platform (Alkabbany et al., 2023). The implementation is robust and handles sensor interruptions, missing pose detections, concurrent data recording, and user interface stability through established software design practices. All session data are stored in open and structured formats that support reproducibility and further analysis (Romero & Ventura, 2010; Stodden et al., 2016).

A key interpretive constraint must be acknowledged. The Focus Index measures screen directed behavioral engagement and reflects reduced scores during robot interaction phases regardless of actual task engagement. This is an inherent property of gaze based behavioral proxies in mixed modality learning environments and does not represent a limitation of the implementation itself. It is addressed through the three signal disengagement rule and requires contextual interpretation by teachers and researchers.

6.2. Future Work

The current system establishes a complete and functional foundation for sensor based engagement assessment in educational robotics classrooms. Several directions for future work are identified.

First, formal validation of the Focus Index is required. The current signal weights are based on prior literature but have not been calibrated against human annotated engagement data in the ALPER-EU context. A structured validation study comparing system outputs with observer annotations would allow quantitative evaluation and refinement of the model (D'Mello et al., 2017; Sümer et al., 2021).

Second, activity phase tagging would improve interpretability. Currently, score reductions during robot interaction phases cannot be automatically distinguished from disengagement without contextual knowledge. Introducing a tagging mechanism for robot interaction periods would allow analysis outputs to separate screen directed attention from hands on activity.

Third, machine learning based engagement inference could be explored once sufficient annotated data are available. A supervised model trained on multimodal features could capture more complex behavioral patterns and adapt to individual differences, complementing the current rule based approach (Booth et al., 2023; Dewan et al., 2019).

Fourth, personalized calibration could improve signal accuracy. A short calibration phase at the beginning of each session could establish a baseline posture for each student, allowing distance and orientation measures to be adjusted to individual characteristics.

Fifth, multi student monitoring represents an important extension. Expanding the system to track multiple students simultaneously would increase its practical value in classroom environments and require updates to both sensing and data association components (Asteriadis et al., 2013; Raca & Dillenbourg, 2013).

Sixth, real time engagement feedback could support live teaching. Computing the Focus Index during recording and displaying it through the interface would provide immediate awareness of engagement trends.

Seventh, longitudinal analysis across sessions should be explored. Applying the system across multiple ALPER-EU sessions would allow investigation of how engagement evolves over time and how different activities influence student behavior (Romero & Ventura, 2010; Siemens & Baker, 2012).

Eighth, affective state detection represents a meaningful extension of the current behavioral framework. The system currently captures observable behavioral signals but does not attempt to infer emotional or affective states. Future work could extend the sensing pipeline to include facial expression analysis and physiological signals, enabling the detection of affective states such as frustration, boredom, confusion, and excitement during learning activities. Integrating affective signals alongside behavioral engagement indicators would provide a richer and more complete representation of the student experience during educational robotics sessions (D'Mello et al., 2017; Monkaresi et al., 2017).

Finally, clinical and neurodevelopmental applications represent a longer term research direction. Students with neurodevelopmental conditions such as Attention Deficit Hyperactivity Disorder (ADHD) or Autism Spectrum Disorder (ASD) often exhibit distinct and consistent behavioral patterns during structured learning tasks, including differences in gaze duration, attention shift frequency, and body movement characteristics. The non intrusive nature of the current system makes it a potential candidate for exploration as a passive behavioral screening tool, where engagement profiles collected during normal classroom activity could provide indicators that warrant further specialist assessment. This direction requires careful ethical consideration and close collaboration with clinical professionals, and is therefore identified as a long term research direction rather than an immediate system extension (American Psychiatric Association, 2013; Goldberg et al., 2021).

7. Appendices

7.1. Appendix A: Installation Manual

A note on terminal commands

Several steps in this manual require typing commands into the Windows PowerShell terminal. To open PowerShell at any point, press the Windows key, type PowerShell, and press Enter. All commands shown in this manual must be entered in the PowerShell terminal, not in a browser address bar, Windows search bar, or File Explorer.

7.1.1. Development and Testing Environment

The system was developed and tested on the following verified configuration. All installation steps described in this appendix were confirmed on this environment as well as on two additional clean Windows machines.

Table 14 - Development Environment

Component	Details
Operating System	Windows 11 Pro, Version 23H2, Build 22631.6199
Python	3.11.9
Git	2.47.1
Tobii Eye Tracking Software	2.16.8.214
Eye Tracker	Tobii Eye Tracker 4C
Depth Camera	Intel RealSense D455

7.1.2. Hardware Prerequisites

Both sensors must be connected via USB 3.0 ports before launching the application. It is recommended to connect each sensor to a separate USB port to ensure sufficient bandwidth for simultaneous operation.

Tobii Eye Tracker 4C placement
The eye tracker must be mounted at the centre of the lower edge of the monitor bezel, not on top of the screen. Mounting it off centre or above the screen will reduce tracking accuracy. The device ships with a flex mount and metal plate mount. To attach the metal plate mount, remove the protective strip from the adhesive pad and press it firmly against the centre of the bottom bezel. Clean the screen surface before attaching to ensure the adhesive holds correctly. Allow the adhesive to bond for several hours before placing the tracker on the mount. The recommended distance between the tracker and the user is 55 to 95 cm.

Intel RealSense D455 placement
The camera should be mounted on a desktop tripod and positioned beside the monitor at desk level, angled to face the student workspace directly. The camera must have a clear and unobstructed view of the student from the front. Avoid placing it at extreme angles as this affects the accuracy of body landmark detection and distance estimation.

7.1.3. Step 1 Install Python 3.11.9

Download Python 3.11.9 from:

https://www.python.org/downloads/release/python-3119/

Scroll to the Files section and select Windows installer 64 bit. Run the installer. On the first screen, check the box labelled "Add Python 3.11 to PATH" before clicking Install Now. This step is mandatory. If this box is not checked, the py command will not be recognised in the terminal.

To verify the installation, open PowerShell and run:

```
py -3.11 --version
```

Expected output: Python 3.11.9

If the machine already has other Python versions installed, use the Windows Python Launcher with `py -3.11` to ensure the correct interpreter is used.

7.1.4. Step 2 Configure PowerShell Execution Policy

Run the following command once in PowerShell:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

Type Y when prompted. This setting applies only to the current user.

7.1.5. Step 3 Install Git

Download Git from:

https://git-scm.com/download/win

Run the installer with default options. After installation, reopen PowerShell and verify:

```
git --version
```

Expected output: git version 2.x.x.windows.x

7.1.6. Step 4 Clone the Repository

```
git clone https://github.com/daveak17/ALPER-EU.git

cd ALPER-EU
```

7.1.7. Step 5 Install Python Dependencies

```
py -3.11 -m pip install -r requirements.txt
```

This installs mediapipe, pandas, scipy, matplotlib, pyrealsense2, pyzmq, openpyxl, Pillow and dependencies.

Note: pyrealsense2 includes the full Intel RealSense runtime.
Note: numpy and opencv are installed automatically as dependencies of mediapipe.

Verify installation:

```
py -3.11 -c "import mediapipe; import pyrealsense2; import zmq; print('OK')"
```

Expected output: OK

7.1.8. Step 6 Install Tobii Eye Tracking Software

Download Tobii Eye Tracking Core Software version 2.16.8 from:

https://gaming.tobii.com/getstarted/

On the page select Tobii → Tobii Eye Tracker 4C → Tobii Eye Tracking Core Software.

The installer filename is:

```
Tobii_Eye_Tracking_Core_v2.16.8.214_x86.exe
```

This installer includes the Tobii Eye Tracking application, USB device drivers, and the Stream Engine service. No additional downloads are required.

Compatibility note:

The Tobii download page states that this software is not officially supported on Windows 11. The system was successfully tested on Windows 11 Pro Version 23H2 and operates correctly despite this warning.

After installation, plug in the Tobii Eye Tracker 4C and complete the following steps:

- Open the Tobii Eye Tracking application from the Windows system tray
- Create a user profile when prompted
- Complete the eye tracking calibration

TobiiStream will not produce valid gaze data until a calibrated user profile exists.

7.1.9. Step 7 Launch TobiiStream

Path:

```
Required/TobiiStream/TobiiStream/TobiiStream.exe
```

Run the executable. A console window will appear.

Expected message:

```
SUCCESS: The connection to the GazeTrack Processing library has been correctly
set up!
```

Keep this console window open throughout all recording sessions. Closing it will interrupt the gaze data stream.

7.1.10. Step 8 Launch the Application

```
py -3.11 MainApp.py
```

On successful startup the application window opens with three tabs. Within a few seconds of startup both the Eye tracker and Body tracker indicators should show Connected in green, confirming that both sensor pipelines have been detected and the system is ready to record.

7.2. Appendix B: User Manual

7.2.1. System Overview

The ALPER-EU Multimodal Engagement Recorder is a desktop application for teachers facilitating educational robotics sessions. It records synchronized gaze and body tracking data using a Tobii Eye Tracker 4C and an Intel RealSense D455.

The system computes a Focus Index representing screen directed behavioral engagement and produces CSV datasets, a synchronized video recording, and a teacher facing Excel report.

7.2.2. Before Starting a Session

Confirm:

- Sensors connected

- The Tobii Eye Tracking application is running in the Windows system tray
- TobiiStream.exe is running and the console window is open
- Application launched with `py -3.11 MainApp.py` via PowerShell
- Both sensors show Connected

7.2.3. Session Control Tab

Countdown timer

Set between 0 and 60 seconds. Recording starts automatically after countdown.

Recording buttons

- Start BOTH
- Start Eye Only
- Start Body Only

When Start BOTH is clicked, a metadata dialog appears. You can press Guest or Skip.

Live Engagement Monitor

- Gaze on screen
- Within distance
- Facing forward
- Session time
- Student engaged

Student is disengaged only when all three signals fail.

7.2.4. During a Session

No manual input required.

Live Tracking tab shows video and engagement score.

7.2.5. Stopping a Session

Click Stop BOTH.

System:

- Saves CSV files
- Finalises video
- Displays verification panel

Files saved in:

```
recordings/both_YYYYMMDD_HHMMSS/
```

7.2.6. Running Post Session Analysis

Go to Data Analysis tab.

Options:

- Use Last Session
- Browse

Click Run Full Analysis.

Table 15 - Analysis Outputs

Output	Description
Engagement Timeline	Per second engagement bands
Gaze Heatmap	Gaze distribution
Signal Summary	Signal activity percentages
Engagement Score	Smoothed Focus Index

7.2.7. Downloading the Session Report

Click Download Session Report.

Exports Excel file with multiple sheets.

Excel contains:

- Session Report
- Engagement Summary
- Gaze Data
- Body Data
- Disengagement Events

7.2.8. Multi Session Comparison

- Add Session
- Run Comparison

7.2.9. Session Output Files Reference

Table 16 - Output Files

File	Contents
`tobii_gaze.csv`	Gaze data
`upper_body.csv`	Pose and signals
`hands_data.csv`	Hand detection
`realsense_color.avi`	Video recording
`session_metadata.json`	Session metadata
`analysis/session_summary.json`	Metrics
`analysis/*.png`	Plots

7.3. Appendix C: Hardware Substitution Guide

7.3.1. Overview

This appendix describes how to substitute the Tobii Eye Tracker 4C and the Intel RealSense D455 with alternative hardware. It is intended for researchers who wish to deploy the

system using different sensing devices. While the architectural interface requirements are defined in the Implementation Phase, this appendix focuses on the practical steps required to perform the substitution.

7.3.2. Substituting the Eye Tracker

The Python application does not communicate directly with the eye tracker. Instead, it subscribes to a local ZeroMQ socket and receives preprocessed gaze messages from a bridge process. To substitute a different eye tracker, a new bridge must be implemented that reads data from the device and publishes messages in the required format.

Required message format:

```
TobiiStream <sequence> <raw_x> <raw_y> <validity>
```

The bridge must publish messages to:

```
tcp://127.0.0.1:5556
```

A minimal Python publisher implementation is shown below:

```
import zmq

context = zmq.Context()
socket = context.socket(zmq.PUB)
socket.bind("tcp://127.0.0.1:5556")

# In the device reading loop:
socket.send_string(f"TobiiStream {seq} {x} {y} {validity}")
```

Tobii Eye Tracker 5

The Tobii Eye Tracker 5 uses the same Tobii Eye Tracking Core Software stack. A bridge targeting the Tobii 5 SDK can publish messages in the same format, requiring no changes to the Python application. The device is also officially supported on Windows 11, which improves compatibility compared to the 4C.

Non Tobii eye trackers

For other devices, a bridge must be implemented using the vendor SDK. The bridge can be written in any language, provided it publishes ZeroMQ messages in the required format. The bridge must be running before `MainApp.py` is launched, in the same way that `TobiiStream.exe` is used in the current setup.

Practical requirements:

- Gaze coordinates must be provided in screen pixels at the configured resolution (default 1920 × 1200)
- The device must provide a stable output rate of at least 30 Hz
- If the display resolution differs, update `SCREEN_W` and `SCREEN_H` in `engagement_analysis.py` and the normalization bounds in `GazeAppAlpha.py`
- The device should be mounted centrally at the bottom bezel of the monitor at a distance of 55 to 95 cm from the user

Dependency changes:

- `pyzmq` remains required
- Replace TobiiStream.exe with the new bridge process
- No other dependencies need to be modified

7.3.3. Substituting the Depth Camera

Alternative Intel RealSense devices such as D435, D415, and D435i

These devices use the same `pyrealsense2` API and pipeline configuration. No code changes are required. Verify that the device supports the configured streams:

- Depth: 848 × 480 at 30 FPS
- Color: 1280 × 720 at 30 FPS

Non RealSense depth cameras such as Azure Kinect, OAK D, and Zed 2

To use a different depth camera, replace the `pyrealsense2` initialization block located at the top of `_preview_loop()` in `body-tracker.py`. The block to replace is:

```
pipeline = rs.pipeline()
config = rs.config()
config.enable_stream(rs.stream.depth, 848, 480, rs.format.z16, 30)
config.enable_stream(rs.stream.color, 1280, 720, rs.format.bgr8, 30)
pipeline.start(config)
align = rs.align(rs.stream.color)
```

The replacement implementation must provide:

- A BGR NumPy array called `color_image` at 30 FPS for MediaPipe processing and video recording

- A depth lookup mechanism equivalent to `depth_frame.get_distance(x, y)` for retrieving the distance at a given pixel coordinate

All other components, including MediaPipe inference, orientation detection, CSV writing, and video encoding, remain unchanged.

Practical requirements:

- Must provide synchronized RGB and depth streams at 30 FPS
- RGB resolution of at least 640 × 480, with 1280 × 720 recommended
- Depth data must support per pixel distance queries
- Minimum horizontal field of view of 60 degrees
- Must support stable mounting with a clear frontal view of the user

Dependency changes:

- Remove `pyrealsense2` from requirements.txt
- Add the Python SDK for the replacement device
- All other dependencies such as `mediapipe`, `opencv python`, `numpy`, `pandas`, and `pyzmq` remain unchanged

7.3.4.

Table 17 - Summary of hardware substitution requirements for eye tracking and depth sensing components.

Component	Current Device	Replacement Requirement
Eye tracker	Tobii Eye Tracker 4C	Any device with a bridge publishing TobiiStream at 30 Hz or higher
Eye tracker bridge	TobiiStream.exe	Any process publishing the required ZeroMQ format
Depth camera	Intel RealSense D455	Any device providing BGR frames and per pixel depth at 30 FPS
Gaze dependency	pyzmq	No change
Depth dependency	pyrealsense2	Replace with device SDK
Pose estimation	MediaPipe	No change
Screen resolution	1920 × 1200	Update constants if needed

References

1. Alkabbany, I., Ali, A. M., Foreman, C., Tretter, T., Hindy, N., & Farag, A. (2023). An Experimental Platform for Real-Time Students Engagement Measurements from Video in STEM Classrooms. *Sensors*, *23*(3), 1614. https://doi.org/10.3390/s23031614
2. Altuwairqi, K., Jarraya, S. K., Allinjawi, A., & Hammami, M. (2021). Student behavior analysis to measure engagement levels in online learning environments. *Signal, Image and Video Processing*, *15*(7), 1387–1395. https://doi.org/10.1007/s11760-021-01869-7

3. American Psychiatric Association. (2013). *Diagnostic and Statistical Manual of Mental Disorders*. American Psychiatric Association. https://doi.org/10.1176/appi.books.9780890425596
4. Appel, T., Scharinger, C., Gerjets, P., & Kasneci, E. (2018). Cross-subject workload classification using pupil-related measures. *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications*, 1–8. https://doi.org/10.1145/3204493.3204531
5. Araya, R., & Sossa-Rivera, J. (2021). Automatic Detection of Gaze and Body Orientation in Elementary School Classrooms. *Frontiers in Robotics and AI*, *8*. https://doi.org/10.3389/frobt.2021.729832
6. Asteriadis, S., Chatzitofis, A., Zarpalas, D., Alexiadis, D. S., & Daras, P. (2013). Estimating human motion from multiple Kinect sensors. *Proceedings of the 6th International Conference on Computer Vision / Computer Graphics Collaboration Techniques and Applications*, 1–6. https://doi.org/10.1145/2466715.2466727
7. Beazley, D. (2010). *Understanding the Python GIL. PyCon 2010.* http://www.dabeaz.com/python/UnderstandingGIL.pdf
8. Beazley, D., & Jones, B. K. (2013, May). *Python Cookbook, 3rd Edition*. Https://Learning.Oreilly.Com/Library/View/Python-Cookbook-3rd/9781449357337/.
9. Bidwell J., & Fuchs H. (2011, November). *(PDF) Classroom Analytics: Measuring Student Engagement with Automated Gaze Tracking*. https://www.researchgate.net/publication/301956819_Classroom_Analytics_Measuring_Student_Engagement_with_Automated_Gaze_Tracking?channel=doi&linkId=572cb7f908aee0229759797a&showFulltext=true
10. Booth, B. M., Bosch, N., & D'Mello, S. K. (2023). Engagement Detection and Its Applications in Learning: A Tutorial and Selective Review. *Proceedings of the IEEE*, *111*(10), 1398–1422. https://doi.org/10.1109/JPROC.2023.3309560
11. Bosch, N., D'Mello, S., Baker, R., Ocumpaugh, J., Shute, V., Ventura, M., Wang, L., & Zhao, W. (2015). Automatic Detection of Learning-Centered Affective States in the Wild. *Proceedings of the 20th International Conference on Intelligent User Interfaces*, 379–388. https://doi.org/10.1145/2678025.2701397
12. Bradski, G., & Kaehler, A. (2008). Learning OpenCV. In *https://learning.oreilly.com/library/view/learning-opencv/9780596516130/*.
13. Card, S. K., Moran, T. P., & Newell, A. (2018). *The Psychology of Human-Computer Interaction* (S. K. Card, Ed.). CRC Press. https://doi.org/10.1201/9780203736166

14. Chen, Y., Li, J., Liu, Y., & Jiang, F. (2025). A Novel Student Engagement Analysis of Real Classroom Teaching Using Unified Body Orientation Estimation. *Sensors*, *25*(20), 6421. https://doi.org/10.3390/s25206421
15. Dewan, M. A. A., Murshed, M., & Lin, F. (2019). Engagement detection in online learning: a review. *Smart Learning Environments*, *6*(1), 1. https://doi.org/10.1186/s40561-018-0080-z
16. D'Mello, S., Dieterle, E., & Duckworth, A. (2017). Advanced, Analytic, Automated (AAA) Measurement of Engagement During Learning. *Educational Psychologist*, *52*(2), 104–123. https://doi.org/10.1080/00461520.2017.1281747
17. D'Mello, S., & Graesser, A. (2012). Dynamics of affective states during complex learning. *Learning and Instruction*, *22*(2), 145–157. https://doi.org/10.1016/j.learninstruc.2011.10.001
18. D'Mello, S., Lehman, B., Pekrun, R., & Graesser, A. (2014). Confusion can be beneficial for learning. *Learning and Instruction*, *29*, 153–170. https://doi.org/10.1016/j.learninstruc.2012.05.003
19. Fredricks, J. A., Blumenfeld, P. C., & Paris, A. H. (2004). School Engagement: Potential of the Concept, State of the Evidence. *Review of Educational Research*, *74*(1), 59–109. https://doi.org/10.3102/00346543074001059
20. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software. Addison-Wesley.*
21. Goldberg, P., Sümer, Ö., Stürmer, K., Wagner, W., Göllner, R., Gerjets, P., Kasneci, E., & Trautwein, U. (2021). Attentive or Not? Toward a Machine Learning Approach to Assessing Students' Visible Engagement in Classroom Instruction. *Educational Psychology Review*, *33*(1), 27–49. https://doi.org/10.1007/s10648-019-09514-z
22. Gorelick, M., & Ozsvald, I. (2020). High Performance Python, 2nd Edition. In *https://learning.oreilly.com/library/view/high-performance-python/9781492055013/*.
23. Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2
24. Hintjens, P. (2013). *ZeroMQ: Messaging for many applications.* 1–516.
25. Holmes, W., Porayska-Pomsta, K., Holstein, K., Sutherland, E., Baker, T., Shum, S. B., Santos, O. C., Rodrigo, M. T., Cukurova, M., Bittencourt, I. I., & Koedinger, K. R. (2022). Ethics of AI in Education: Towards a Community-

Wide Framework. *International Journal of Artificial Intelligence in Education*, *32*(3), 504–526. https://doi.org/10.1007/s40593-021-00239-1

26. Hunt, A., & Thomas, D. (1999, October 13). *The pragmatic programmer: From journeyman to master.* Https://Www.Academia.Edu/41066052/Andrew_Hunt_and_David_Thomas_Pragmatic_Programmer.
27. Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55
28. Intel Corporation. (2023). *Intel® RealSense™ product family D400 series datasheet.*
29. Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., Zhang, F., Chang, C.-L., Yong, M. G., Lee, J., Chang, W.-T., Hua, W., Georg, M., & Grundmann, M. (2019). *MediaPipe: A Framework for Building Perception Pipelines*. http://arxiv.org/abs/1906.08172
30. Lundh, F. (1999). *An Introduction to Tkinter*. http://www.pythonware.com/library/tkinter/introduction/
31. Lutz, M. (2013). Learning Python, 5th Edition. In *https://learning.oreilly.com/library/view/learning-python-5th/9781449355722/*.
32. Martelli, Alex., Ravenscroft, A. Martelli., & Ascher, David. (2005). *Python cookbook (2nd ed.).* O'Reilly.
33. McKinney, W. (2010). *Data Structures for Statistical Computing in Python*. 56–61. https://doi.org/10.25080/Majora-92bf1922-00a
34. Monkaresi, H., Bosch, N., Calvo, R. A., & D'Mello, S. K. (2017). Automated Detection of Engagement Using Video-Based Estimation of Facial Expressions and Heart Rate. *IEEE Transactions on Affective Computing*, *8*(1), 15–28. https://doi.org/10.1109/TAFFC.2016.2515084
35. Oliphant, T. E. (2007). Python for Scientific Computing. *Computing in Science & Engineering*, *9*(3), 10–20. https://doi.org/10.1109/MCSE.2007.58
36. Pabba, C., & Kumar, P. (2022). An intelligent system for monitoring students' engagement in large classroom teaching through facial expression recognition. *Expert Systems*, *39*(1). https://doi.org/10.1111/exsy.12839
37. Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, *15*(12), 1053–1058. https://doi.org/10.1145/361598.361623
38. Perrotta, C., & Selwyn, N. (2020). Deep learning goes to school: toward a relational understanding of AI in education. *Learning, Media and Technology*, *45*(3), 251–269. https://doi.org/10.1080/17439884.2020.1686017
39. Prieto, L. P., Sharma, K., Dillenbourg, P., & Jesús, M. (2016). Teaching analytics. *Proceedings of the Sixth International Conference on Learning*

Analytics & Knowledge - LAK '16, 148–157. https://doi.org/10.1145/2883851.2883927

40. Python Software Foundation. (2024). *Python documentation (Version 3)*. https://docs.python.org/3/
41. Raca, M., & Dillenbourg, P. (2013). System for assessing classroom attention. *Proceedings of the Third International Conference on Learning Analytics and Knowledge*, 265–269. https://doi.org/10.1145/2460296.2460351
42. Reyes, M. R., Brackett, M. A., Rivers, S. E., White, M., & Salovey, P. (2012). Classroom emotional climate, student engagement, and academic achievement. *Journal of Educational Psychology*, *104*(3), 700–712. https://doi.org/10.1037/a0027268
43. Romero, C., & Ventura, S. (2010). Educational Data Mining: A Review of the State of the Art. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *40*(6), 601–618. https://doi.org/10.1109/TSMCC.2010.2053532
44. Shipman, J. W. (2013). *Tkinter 8.5 reference: a GUI for Python*. http://www.nmt.edu/tcc/help/pubs/tkinter/tkinter.pdf
45. Siemens, G., & Baker, R. S. J. d. (2012). Learning analytics and educational data mining. *Proceedings of the 2nd International Conference on Learning Analytics and Knowledge*, 252–254. https://doi.org/10.1145/2330601.2330661
46. Stodden, V., McNutt, M., Bailey, D. H., Deelman, E., Gil, Y., Hanson, B., Heroux, M. A., Ioannidis, J. P. A., & Taufer, M. (2016). Enhancing reproducibility for computational methods. *Science*, *354*(6317), 1240–1241. https://doi.org/10.1126/science.aah6168
47. Sümer, Ö., Goldberg, P., D'Mello, S., Gerjets, P., Trautwein, U., & Kasneci, E. (2021). Multimodal Engagement Analysis From Facial Videos in the Classroom. *IEEE Transactions on Affective Computing*, *14*(2), 1012–1027. https://doi.org/10.1109/TAFFC.2021.3127692
48. Tanenbaum, A. S., & Van Steen, M. (2007). *Distributed systems: Principles and paradigms (2nd ed.). Prentice Hall.* www.minix3.org.
49. Tobii AB. (2021). *Specifications for the Tobii Eye Tracker 4C*. https://help.tobii.com/hc/en-us/articles/213414285-Specifications-for-the-Tobii-Eye-Tracker-4C
50. Whitehill, J., Serpell, Z., Lin, Y.-C., Foster, A., & Movellan, J. R. (2014). The Faces of Engagement: Automatic Recognition of Student Engagement from Facial Expressions. *IEEE Transactions on Affective Computing*, *5*(1), 86–98. https://doi.org/10.1109/TAFFC.2014.2316163

51. Zaletelj, J., & Košir, A. (2017). Predicting students' attention in the classroom from Kinect facial and body features. *Eurasip Journal on Image and Video Processing*, *2017*(1). https://doi.org/10.1186/s13640-017-0228-8