Politecnico di Milano

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING

DIGITAL FACTORY – Prof. Marcello Urgo

An Example of Distraction Detection in Manufacturing Systems using Project Aria Glasses

Supervisor(s)

Marcello Urgo

Walter Terkaj

Students

Filippo Zavatta – 10772831

Giorgio Colella – 10744588

Lorenzo Ravasio – 10793207

Academic Year 2024 – 2025

Index

1. Introduction 4

1.1. Human Monitoring 4

1.2. Project Aria 5

1.3. Neural Networks 6

1.4. Problem statement 7

1.5. General Objective 8

2. Approach 9

2.1. Core Idea 9

2.2. Workplan 10

2.3. Disassembly process study 11

2.4. Project boundaries 12

2.5. Experiment definition 12

3. Aria Glasses 14

3.1. Hardware 14

3.2. Visual sensors 14

3.3. Non-Visual sensors 14

3.4. Software and applications 15

3.5. MPS 15

3.6. VRS and MPS ‘outputs data format 16

3.7. Aria Project python library 17

3.8. Coordinate conventions 18

4. Additional Tools 20

4.1. Google Colab 20

4.2. Yolo v11 Neural Network 20

4.3. Roboflow 20

4.4. Libraries *20*

5. Work Phases **22**

5.1. Aria glasses exploration *22*

5.2. Pictures labelling *23*

5.3. Algorithm training *24*

5.4. Eye Gaze coordinates conversion *27*

5.5. Inference on frames *28*

5.6. Testing of eye gaze inside bounding box *28*

5.7. Video processing *29*

5.8. Notebook "test_video" *29*

6. Experiment and results **31**

6.1. Choice of confidence level for our algorithm *31*

6.2. Experiment *31*

6.3. Results *32*

6.4. Example of Proof of Concept *32*

7. Future developments **33**

8. Conclusions **34**

Annex 1 .... **35**

References **36**

1. Introduction

The problem we are facing for this project falls within the scope of human modelling and monitoring, as we were asked to develop a system capable of **distraction detection in a manufacturing environment**. This approach holds multiple advantages for companies, ranging from a reduction in costs associated with errors, related to higher activity accuracy, to an expected improvement in productivity. Our work revolves around the following key elements: **human monitoring activities** in manufacturing, **Meta's Aria Project** and the application of **neural networks.** In this introductory chapter, we will provide a brief overview of the topics mentioned above, followed by a definition of the specific problem we focused on and the project's general objective.

1.1. Human Monitoring

The last decades have seen an increase in the variability that factories must face and in environmental changes. This condition led to a drastic change of production management paradigm that passed from mass production to flexible production to face the increasing uncertainty. As reported by [1]:

> *"In industrial sectors where the pace of change is quite fast, we believe that the era of mass production ended in the 1980s with the emergence of flexible/agile production as the dominant paradigm. We are not declaring the "end" of mass production as a production process but rather as the dominant paradigm of industrial management"*

Instead, other refer to flexibility defined as the "*…ability to adapt rapidly and with constant co-ordination in an environment of constant and rapid change." [2].* To achieve this goal *Industry 4.0* paradigm has introduced several technologies such as Additive Manufacturing, Internet of Things or Cloud Computing. But there is one element that will remain crucial along the fourth industrial revolution, and it is the **human factor.**

Human operators will be fundamental to guarantee the necessary flexibility that machines cannot achieve. A big goal for the future is the development of human-centred systems where machines and humans collaborate.

The presence of humans is necessary, but it also introduces variability and safety issues in the production. In the future, it will be crucial to exploit new technologies to limit this variability ensuring a steady production rate and avoiding quality issues. A great help for the achievement

of this goal comes from AI, in particular **Computer Vision**. The use of Computer Vision algorithms in the factories, accompanied by widespread cameras installation, will enable the control of human operators checking if the operations stages are followed correctly.
The vast majority of human errors in factories comes from distractions and the widespread diffusion of digital devices is worsening the situation. As stated by Jim Phelan in The Cost of Digital Distraction in Industrial Environments (2021):

> *"According to a nation-wide survey of 1,769 full-time US employees, 87% of workers said that digital distraction hurts their organization's overall productivity. It's well known that smart phones can be very addicting, and while many people have acquired the habit of checking their smartphones and scrolling through their social media feeds throughout the day, this can have serious implications in an industrial environment." [1]*

And continues:

> *"While distraction causes productivity decreases for most organizations, in these industrial environments the stakes are much higher. In these workplaces, the risk of distraction causing accidents that can lead to serious injuries and property damage, increase significantly." [2]*

Finding smart ways to reduce distractions will be a key challenge in the future of manufacturing. Reliable and accurate distraction detection systems can lead to radical improvement in terms of productivity and operators' safety.
Monitoring systems rely on multiple cameras recording all the process stages from different perspectives. For distraction detection, egocentric monitoring systems will play a crucial role. These are systems that take the perspective of the operator and monitor both the external environment and the operator actions from a closer point of view. An example of egocentric monitoring system is represented by the Project Aria glasses that we had the opportunity to lever for our project.

1.2. Project Aria

For this project, we will leverage the Project Aria glasses supplied to Politecnico di Milano by Meta Platforms.
Meta Platforms Inc., commonly referred to as Meta, is a leading U.S.-based technology company headquartered in Menlo Park, California. Established in 2004 as Facebook, Inc., the company rebranded to its current name on October 28, 2021. Meta owns and operates prominent

platforms such as Facebook, Instagram, WhatsApp, and Messenger and develops cutting-edge virtual reality devices, including the Oculus Rift. Meta's mission extends beyond traditional social networks to create immersive experiences through augmented, virtual, and mixed reality, driven by principles of community connection, privacy protection, and fostering economic opportunities.

Project Aria, an initiative by Meta Reality Labs Research, is a research platform aimed at advancing egocentric AI across domains like computer vision, robotics, and contextual AI. It enables collaboration between Meta's internal teams and global research communities to address significant challenges in egocentric AI. Project Aria encompasses two primary components [3]:

1. **Open Science Initiatives**: These provide open-access egocentric datasets, pre-trained AI models, and tools for the research community. Researchers are also encouraged to engage with challenges proposed by Meta to solve specific AI problems.
2. **Aria Research Kit**: This specialized toolkit includes Aria glasses, for multimodal egocentric data collection, a companion mobile app, dataset management applications, post-processing services, and a Client SDK for custom application development. Academic and commercial partners can apply to access this kit.

Meta collaborates with global partners to expand the application of Project Aria. For instance, collaboration with Carnegie Mellon University's (CMU) NavCog project leverages Project Aria to build 3D maps of indoor spaces like airports and museums, enabling better navigation for visually impaired individuals.

1.3. Neural Networks

Neural networks, inspired by the interconnected structure of the human brain, are a foundational tool in modern machine learning. Developed in the mid-20th century, their evolution was spurred by advancements in computational power and algorithms, enabling machines to recognize patterns and learn from data. At their core, neural networks consist of layers of interconnected nodes (neurons), which process input data, assign weights to features, and adjust these weights through iterative learning to produce accurate outputs. This mechanism allows them to tackle complex problems like image recognition, natural language processing, and predictive analytics [4].

Building on this foundation, Convolutional Neural Networks (CNNs) extend the functionality of traditional neural networks by specializing in grid-like data, such as images. CNNs

incorporate convolutional layers, which apply filters to the input data to extract hierarchical features, from simple edges to complex patterns. This architecture makes them exceptionally powerful for image and video analysis, where spatial relationships in the data are crucial [5].
In the realm of human monitoring, CNNs have demonstrated significant potential. By analyzing visual data, they can recognize human activities, monitor behaviours, and detect anomalies.

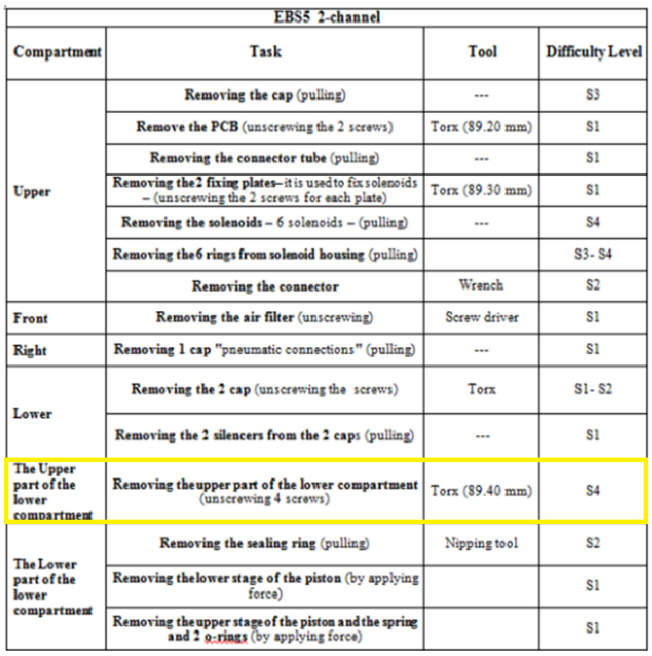
> "*In recent days, applications of HAR in numerous domains such as health monitoring systems, surveillance in industries, traffic control systems, military surveillance, robotics navigation systems, etc. are growing rapidly due to technological advancement in CNN*" [6]

1.4. Problem statement

The process to be monitored involved the disassembly of an EBS5 2-channel, a key component truck's braking system. Figure 1 outlines that the disassembly needs the following tools to be properly executed:

- Torx: a 6-pointed screwdriver bit specific for screws with hexagonal shaped head.
- Wrench: necessary to remove the connector.
- Screwdriver: for the sake of simplicity, we considered the torx as part of the screwdriver when training the algorithm.
- Nipping tool: used to remove the sealing ring for its pulling strength.

For each task you either need one tool or none.
Our experiments focused just on task number 12 that involves the unscrewing of four screws, consequently the main components involved in the process were: the "body" of the component, the screws, and the screwdriver equipped with the torx.



EBS5 2-channel			
Compartment	**Task**	**Tool**	**Difficulty Level**
Upper	**Removing the cap** (pulling)	---	S3
	Remove the PCB (unscrewing the 2 screws)	Torx (89.20 mm)	S1
	Removing the connector tube (pulling)	---	S1
	Removing the2 fixing plates–it is used to fix solenoids – (unscrewing the 2 screws for each plate)	Torx (89.30 mm)	S1
	Removing the solenoids – 6 solenoids – (pulling)	---	S4
	Removing the6 rings from solenoid housing (pulling)		S3- S4
	Removing the connector	Wrench	S2
Front	**Removing the air filter** (unscrewing)	Screw driver	S1
Right	**Removing 1 cap** "pneumatic connections" (pulling)	---	S1
Lower	**Removing the 2 cap** (unscrewing the screws)	Torx	S1- S2
	Removing the 2 silencers from the 2 caps (pulling)	---	S1
The Upper part of the lower compartment	**Removing the upper part of the lower compartment** (unscrewing 4 screws)	Torx (89.40 mm)	S4
The Lower part of the lower compartment	**Removing the sealing ring** (pulling)	Nipping tool	S2
	Removing the lower stage of the piston (by applying force)		S1
	Removing the upper stage of the piston and the spring and 2 o-rings (by applying force)		S1

Figure 1: Project description

1.5. General Objective

This project is centered around a primary objective: demonstrating the potential utility of ego-centric monitoring system applications, such as the Project Aria Glasses, that exploits the neural network potential to perform distraction detection within industrial environments. To achieve this goal, a structured workplan was developed to design a human monitoring system capable of detecting operators' distractions tailored for the brake component disassembly operation. Finally, a proof of concept was created to validate the approach for a specific segment of the process.

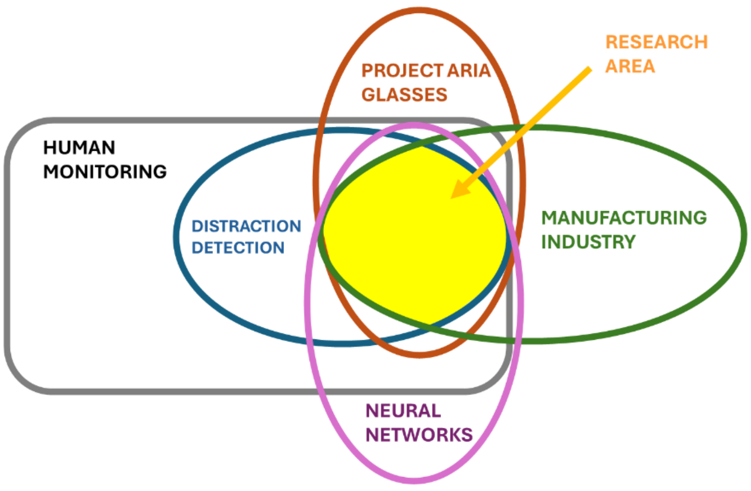


Figure 2: Project scope

2. Approach

This chapter introduce the main idea that guided all our work. We generated a workplan with all the activities needed for the implementation of the core idea and then we selected the activities that we wanted to focus on for this project defining the project boundaries. We then defined an experiment for the algorithm performance evaluation.

2.1. Core Idea

Having in mind the project core objective we identified 3 key **specific objectives** corresponding to information that an algorithm must be able to obtain and process to perform a proper distraction detection:

1. **Eye Gaze direction:** a 2d coordinate corresponding to the point where the operator is looking for each time frame.
2. **Pieces and tools position:** boxes that identify the position of the piece components and the tools used in the disassembly process for each video frame.
3. **Process stage:** a variable that provides an estimation for the actual process stage.

Based on the hypothesis that the operator is focused on the process when they are looking at the components involved in the current operation, the algorithm will detect a distraction whenever this condition is not met. The process stage variable will indicate which objects the operator should look at and, if the Eye gaze direction is inside the bounding boxes of these objects, the operator will be considered focused on the process, otherwise distracted.

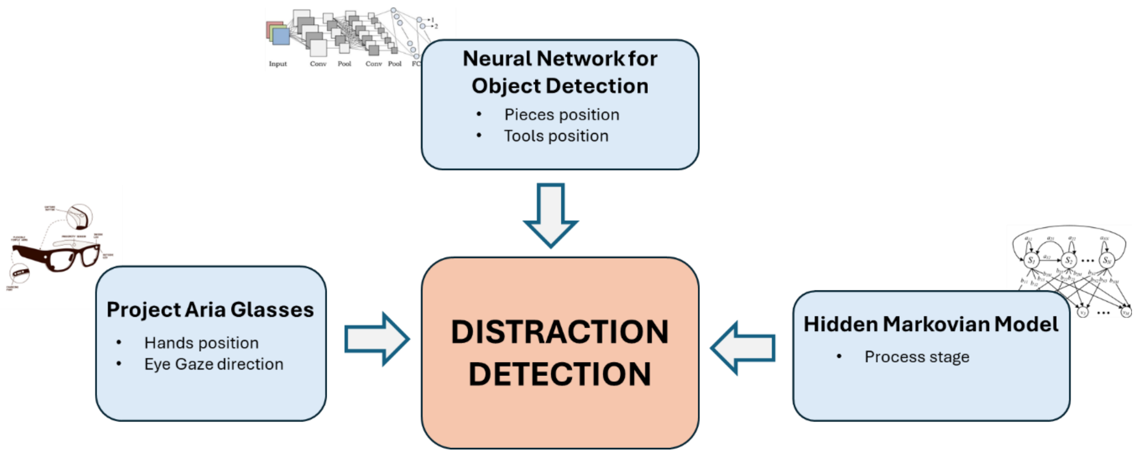


Figure 3: Distraction detection objectives

At this point we identified the tools capable of extracting this four information from a video.

Information about the **Eye gaze direction** and about hands position can be derived from the Project Aria **MPS outputs**, these services provide spatial position data that must be converted into two-dimensional coordinates.

Bounding boxes of the objects can be obtained using a **Convolutional Neural Network** trained on our specific components and tools.

Lastly, an estimation of the **process stage** can be obtained from a **Hidden Markov Model** trained on our specific disassembly process.

2.2. Workplan

The work must start with a study phase to have a clear understanding of the process. During this phase, an Activities on Nodes graph and a model for stage recognition must be produced, highlighting the emissions for the Hidden Markov Model. After this initial phase the work can be split into three independent streams:

MPS output conversion:

Writing the code to convert the three-dimensional MPS output into two-dimensional coordinates.

Convolutional Neural Network training:

1. Components and tools pictures collection.
2. Pre-trained neural network for object detection choice.
3. Pictures labelling.
4. Data augmentation.
5. Division of the whole dataset into training, validation and test set.
6. Training, validation and testing of the neural network.

Hidden Markovian Model training:

1. Define a set of emission.
2. Collecting data about the process, including emission values for each process stage.
3. Hidden Markov Model training and testing with python libraries like Pomegranate.

Once these steps are completed, the work can proceed with the development of the code required to integrate the outputs of the three elements: the MPS, the Neural Network for object detection, and the HMM for stage recognition.

At this point it will be possible to proceed with the final testing conducting an experiment. For the experiment, success criteria will be defined, videos will be recorded, and the algorithm will be tested on the recorded videos. Its performance will be then evaluated.

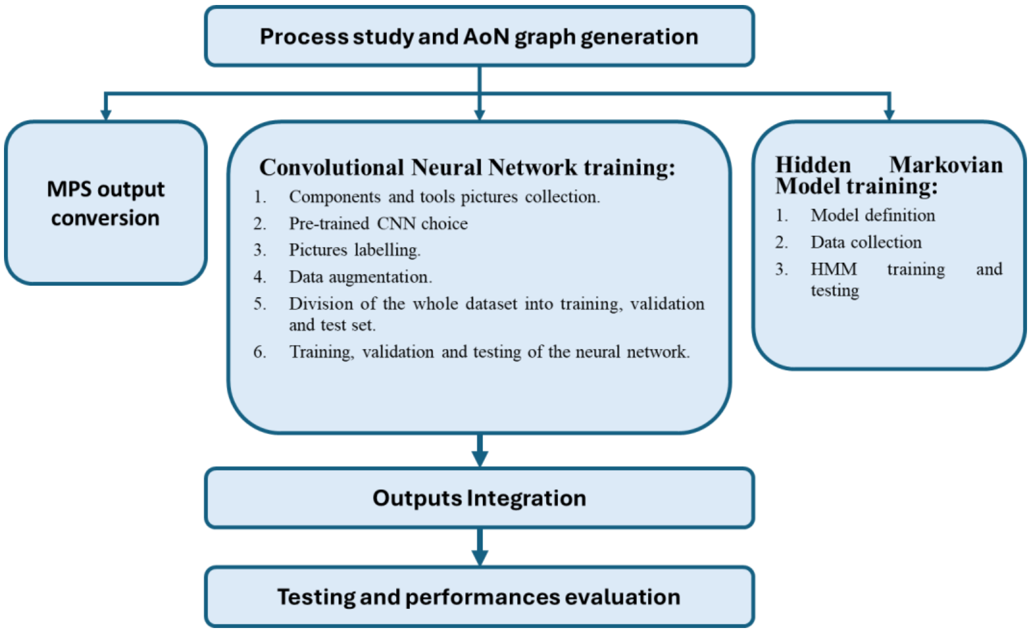

Process study and AoN graph generation
MPS output conversion
Convolutional Neural Network training:
1. Components and tools pictures collection.
2. Pre-trained CNN choice
3. Pictures labelling.
4. Data augmentation.
5. Division of the whole dataset into training, validation and test set.
6. Training, validation and testing of the neural network.
Hidden Markovian Model training:
1. Model definition
2. Data collection
3. HMM training and testing
Outputs Integration
Testing and performances evaluation

Figure 4: Project activities

2.3. Disassembly process study

Based on the activity's descriptions in figure 1, we derived the precedence relations and listed them below, then we created the Activities on Nodes graph depicted in figure 5.

	START	IPA
1	Remove the cap	S
2	Remove the PCB	1
3	Remove the connector tube	2
4	Removing the 2 fixing plates	3
5	Removing 6 solenoids	4
6	Removing 6 rings	5
7	Removing the connector	6
8	Removing the air filter	S
9	Removing 1 cap	S
10	Removing 2 cap	S
11	Removing the 2 silencers	10
12	Removing the upper part of the lower compartment	S
13	Removing the sealing ring	11, 12
14	Removing the lower stage of the piston	13

15	Removing the upper stage of the piston, the spring and 2 rings.	14
	FINISH	7, 8, 9, 15

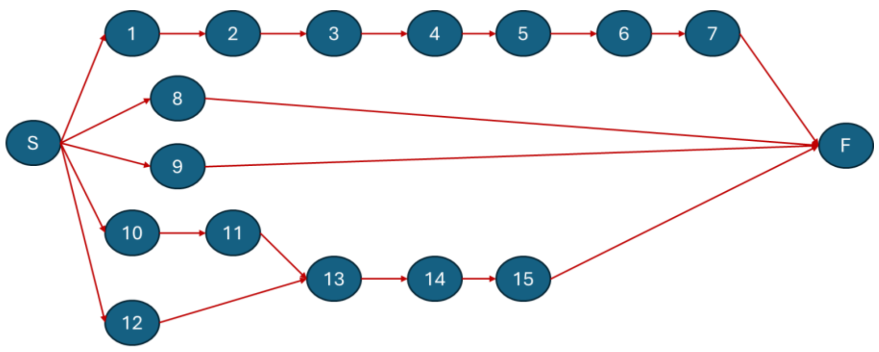


Figure 5: AoN

2.4. Project boundaries

For the development of our Proof of Concept we decided to focus our effort on one specific activity, rather than the whole process. We chose the activity number 12. In this activity the 4 screws that secure the two parts of the lower compartment are unscrewed to separate the two components. The only tool needed for this activity is the 89.4 mm Torx.

Not having sufficient data to train the Hidden Markovian Model for the stage recognition, we decided to take the process stage variable as input.

Additionally, given computational power and connectivity limitations, the proof of concept has been developed on batch data, excluding real time data processing.

2.5. Experiment definition

The experiment is designed to evaluate the performance of the software in two critical areas: the accuracy of object recognition within bounding boxes and the precision of eye gaze alignment with identified objects.

Object Recognition Accuracy: to test our solution’s ability to correctly recognize objects, a video dataset is prepared where various tools (e.g., screws, brakes) are sequentially presented and rotated to expose their front, back, and general views. The software's performance will be evaluated by comparing the detected bounding boxes and the labels with reality. We therefore defined the following metric:

$$Object\ Recognition\ Accuracy = \frac{Number\ of\ correctly\ identified\ frames}{Number\ of\ frames} * 100$$

This metric reflects the system's ability to both locate the objects and correctly classify their label under dynamic conditions.

Eye Gaze Precision: to assess the precision of the eye gaze tracking system, a second video is used. In this video, a team member is instructed to focus their gaze on specific objects for a defined period, i.e., 5 seconds per object. The sequence of objects programmed to focus on in the testing video is as follows:

1 – yellow ticket ~ 5 seconds.

2 – jar ~ 5 seconds.

3 – pen ~ 5 seconds.

We therefore define the following metric:

$$Eye\ Gaze\ Accuracy = 1 - \frac{Mean\ deviation\ from\ the\ target}{Width\ of\ the\ frame}$$

Distraction detection accuracy: Using these two indices, we can define the primary metric to evaluate the results of this project:

$$Distraction\ detection\ accuracy = Eye\ Gaze\ Accuracy * Object\ Recognition\ Accuracy$$

The section Experiment and results explore the computation of those indexes.

3. Aria Glasses

In this section, we will explore and summarize the main information about Aria Project's Glasses. All information has been taken from Aria Project website [7].

3.1. Hardware

The Project Aria glasses are lightweight and adaptable, weighing only 75 grams. Powered by a Qualcomm SD835 processor, the glasses include 4GB of RAM and 128GB of flash memory (UFS) for storage. The device operates on Android 7.1 and provides a configurable user button and switch, allowing flexibility for various applications and research needs. The Aria glasses are equipped with an array of advanced sensors that form the backbone of seamless operation and precise data capture, serving as a critical foundation for egocentric research and development.

3.2. Visual sensors

Aria glasses embed 5 different cameras: 1 RGB camera placed in the top-left corner of the left eye, 2 SLAM cameras placed symmetrically on both sides of the glasses, and 2 internal cameras pointing at the eyes. The specifications of these cameras appear in the table below.

Camera	**Maximum resolution (pix)**	**Max frame rate (FPS)**	**Nominal frame rate (FPS)**	**Shutter**
SLAM (x2)	640x480	30	10	global
RGB (x1)	2880x2880	30	10	rolling
ET (x2)	640x480	90	10	global

3.3. Non-Visual sensors

The glasses integrate various non-visual sensors, including two inertial measurement units (IMUs) operating at 1000 Hz and 800 Hz, a 10 Hz magnetometer, a 50 Hz barometer, and a seven-channel spatial microphone array recording at 48 kHz, with an optional stereo mode. Additionally, the device features GPS, Wi-Fi, and Bluetooth for localization and connectivity.

All sensors are calibrated and synchronized with nanosecond-level precision. Furthermore, the SLAM and RGB cameras use fisheye lenses, maximizing the field of view and enhancing their applicability for spatial and environmental mapping.

3.4. Software and applications

Project Aria Glasses use is facilitated by a wide range of software tools. The list of applications with a short description is reported below:

- **Aria Studio[1]:** With this tool, the glasses connect to the PC using a USB cable. When connected, the app enables starting a recording, downloading recordings saved in the glasses' memory, and requesting an MPS. Using the Rerun software functionalities, from Aria Studio it is also possible to visualize the MPS outputs.
- **Mobile Companion App:** With this tool, the glasses connect to a mobile device if they are on the same Wi-Fi network. Once connected, the app performs eye calibration and records videos that are then saved on the glasses in VRS format. The application also offers the possibility to choose between different profiles for each recording session; users can either use one of the many predefined profiles or create a custom one. A profile allows the user to use only the cameras and sensors functional to the current needs while turning off all others. It also allows selecting other settings such as the cameras' frame rates (different frame rates can also be assigned to different cameras within the same profile).
- **Rerun:** This software enables the visualization of MPS outputs as well as VRS outputs. It is usable from Aria Studio or via Windows PowerShell.

3.5. MPS

From the Aria Studio App, users request three different video analyses through the Aria Machine Perception Services (MPS). Through these services, the VRS uploads and is processed by Meta's algorithms. The analyses that users request include:

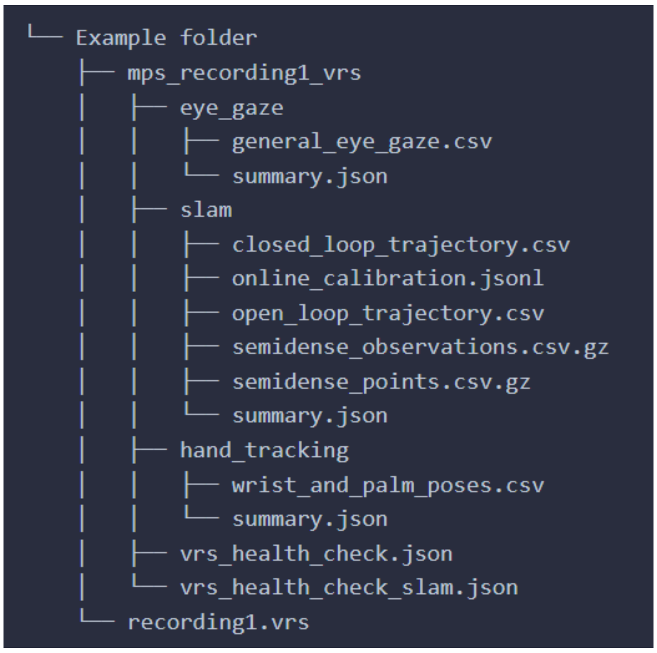
[1] When the project started, Aria Studio were not available for Windows OS, so the **Desktop Companion app** was the only solution available. Meta developers were about to release an update of the whole package of software for the glasses, including a version of Aria Studio. In the beginning of December the Windows version was released, so the team switched to Aria Studio, which has basically the same functions of the Desktop Companion app but with some improvements.

- **Eye Gaze:** Returns information about the glasses' wearer's gaze. From this output, it is possible to know the point in space the wearer is looking at.
- **Hands Tracking:** Returns information about the wearer's hands' position and orientation for each video frame.
- **SLAM**
    - 6DoF trajectory: return info about the device position, speed and acceleration in the space.
    - Semi-dense points cloud: returns the digital 3d reconstruction of the environment in a points cloud format.

The MPS returns as output a json file containing specific information depending on the kind of service requested.

3.6. VRS and MPS 'outputs data format

The recordings are stored inside the glasses' memory and are accessible by logging into the Aria Studio app, where they can be uploaded on the computer. In this section, we describe the two main types of data formats used in this project: VRS and MPS. The first one refers to the format in which Project Aria data is stored, a file format designed and optimized to record and playback streams of XR (extended reality) sensor data and supports huge file sizes, such as images, audio, and other discrete sensors. These VRS files include time-sorted records for each sensor, organized into streams. Each stream contains Configuration, State, and Data records, all timestamped within a common time domain. The sensors in Project Aria glasses align to a shared device-time domain, and multiple devices (such as multiple Aria glasses or Aria glasses with other devices) synchronize using a shared clock. You access VRS data through two main instruments: Python code and the Rerun app. For the first choice, the Aria website offers a wide range of possible requests to access all the possible information provided by the VRSs. On the other hand, the app directly shows on screen all the outputs of the different sensors together with their related information such as time. Of course, the type of data you see on the visualization depends on the profile used to take the recordings, and on the set of sensors that are activated. Machine Perception Services (MPS) are offered as post-processing of VRS files via a cloud service. The MPS services divide into three categories: SLAM, Eye Gaze, and Hand Tracking. Their structure is described by the image below:



```
└── Example folder
    ├── mps_recording1_vrs
    │   ├── eye_gaze
    │   │   ├── general_eye_gaze.csv
    │   │   └── summary.json
    │   ├── slam
    │   │   ├── closed_loop_trajectory.csv
    │   │   ├── online_calibration.jsonl
    │   │   ├── open_loop_trajectory.csv
    │   │   ├── semidense_observations.csv.gz
    │   │   ├── semidense_points.csv.gz
    │   │   └── summary.json
    │   ├── hand_tracking
    │   │   ├── wrist_and_palm_poses.csv
    │   │   └── summary.json
    │   ├── vrs_health_check.json
    │   └── vrs_health_check_slam.json
    └── recording1.vrs
```

Figure 6: MPS folder description

Mps_recording1_vrs is a directory where all the intermediate data and MPS output are saved. It contains information on eye_gaze, slam, and hand_tracking.

The health checks files contain information about issues such as data drops across sensor streams and provide details about some possible health check failures.

The most useful output for our purpose is the eye gaze one, made up of two items: *general_eye_gaze.csv* and *summary.json*. The first consists of a series of variables with their respective values that summarize all the important parameters monitored, for example *tracking_timestamp_us*, contains X, Y, Z coordinates of the right and left eyes relative to the CPF, and the yaw estimation in radians for both right and left eyes. The general_eye_gaze.csv bases itself on the standard eye gaze configuration.

Finally, the item *summary.json* consists of a JSON file containing a high-level report on MPS eye gaze generation, providing an overview of the output for each major stage.

3.7. Aria Project python library

Another Aria Project instrument that proves crucial for our project's scope is the Aria Project python library. Some of the main classes and functions used are:

Aria Data Provider: This API accesses sensor data stored in Project Aria VRS files. It is available in both Python and C++ and allows users to open VRS files and extract data efficiently. Users map labels to stream IDs and vice versa, providing flexible identification of sensor streams. Data is accessed randomly by index or timestamp, supporting multiple time domains such as DEVICE_TIME, recommended for single-device data. The API delivers all sensor data in device-time order through asynchronous iteration.

Data loaders for MPS data: These simplify importing MPS data and formatting it for further analysis. Key functionalities of data loaders for eye gaze MPS analysis include converting yaw and pitch to 3D gaze vectors, re-projecting gaze data onto RGB camera streams, and finding the closest gaze measurement to a given timestamp.

3.8. Coordinate conventions

2D Coordinate Convention: This refers to the convention used for attribute colors to an image: the color value of a pixel with integer coordinates (u,v) is the average color of the square spanning from (u−0.5,v−0.5) to (u+0.5,v+0.5) in continuous coordinates. This definition is important to ensure consistency in visual processing, such as during resizing or interpolation.

3D Coordinate Convention: Every sensor on the Aria glasses operates within its own local coordinate system. The relative pose of each sensor with respect to the "Device Frame" represents itself using a 6-DoF (six degrees of freedom) pose. By default, the Device Frame corresponds to the local frame of the left Mono Scene (SLAM) camera.

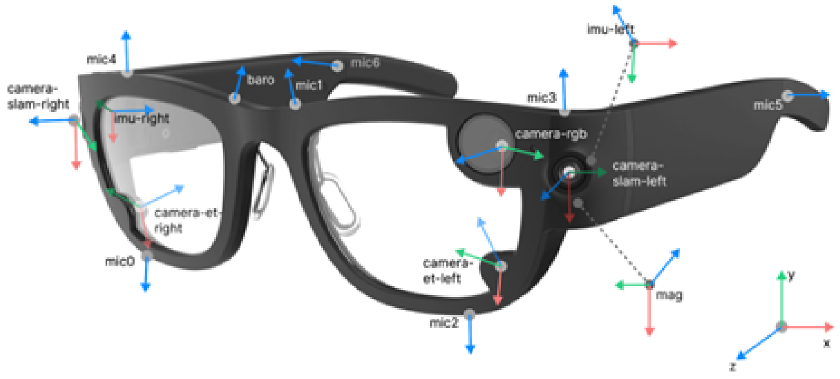


Figure 7: Reference systems of hardware in the glasses

The 6-DoF poses represent themselves using the Lie group SE(3) representation, which combines the (x, y, z) position of an object with its orientation/rotation in space. This results in a 4x4 transformation matrix that integrates both translation and rotation.

Camera Coordinate System: The camera coordinate system places its origin at the optical center, with its axes defined as follows:

- **Z-axis**: Aligned with the optical axis of the lens, pointing forward.
- **X-axis**: Parallel to the projection of the image plane's X-axis onto the pupil.
- **Y-axis**: Calculated as the cross product of the X and Z axes.

This standardized system effectively represents a camera's position and orientation in space.

Non-Visual Sensor Coordinate System: For non-visual sensors, such as the IMU, the coordinate system's origin is located at the position of the accelerometer. The axes are oriented along the accelerometer's sensitive directions and adjusted to correct for sensor orthogonality errors. A similar arrangement applies to the magnetometer.

Central Pupil Frame (CPF): The Central Pupil Frame (CPF) positions itself at the midpoint between the eye boxes of the left and right eyes. In this frame, Aria's Eye-Tracking (ET) gaze defines itself as a vector originating from the (0, 0, 0) point of the CPF.

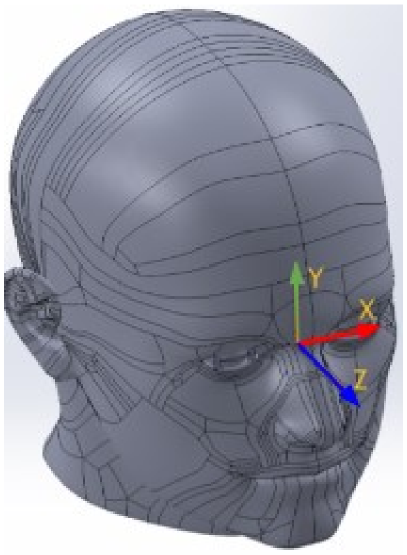


Figure 8: Central Pupil Frame

4. Additional Tools

4.1. Google Colab

We used Google Colab to write and run all the code developed. The platform makes GPUs computational power available for free; this feature is fundamental for the training of the object detection neural network.

4.2. Yolo v11 Neural Network

YOLO v.11, introduced in 2024, is the latest advancement in the YOLO (You Only Look Once) series of real-time object detection models. YOLO is a popular object detection algorithm that revolutionizes the field of computer vision. It is fast and efficient, making it an excellent choice for real-time object detection tasks. It achieves state-of-the-art performance on various benchmarks and is widely adopted in various real-world applications.
Main drivers for the choice of YOLOv11 as the object detection algorithm to train are:

- **Processing speed:** according to [8] *"Methods that use Region Proposal Networks perform multiple iterations for the same image, while YOLO gets away with a single iteration."*. This allows YOLO to outperform other object detection algorithms by a large margin.
- **Ease of use:** YOLO algorithm can be trained, tested, validated and used to make prediction using the Ultralytics library [9]. This library offers user-friendly functions to use the algorithm and to import easily labeled images from the Roboflow platform.

4.3. Roboflow

Roboflow [10] is a platform that offers a space to work on images labelling for object detection algorithms. We chose it for its ease of use. It enables labelling within few clicks and saves images (with the placed box) in a format that is compatible with the YOLO v.11 algorithm.

4.4. Libraries

In this section are listed descriptions of the libraries we used for the development of the code.

Numpy: it's a library for numerical computing, offering efficient operations on multidimensional arrays and matrices. It includes functions for mathematical computations like linear algebra and statistics. Widely used in data science and machine learning, it integrates well with other Python libraries.

OS: the OS module allows interaction with the operating system, providing tools for file and directory management, environment variables, and system-level tasks. It enables the development of cross-platform applications. Key features include path manipulation and process management.

csv: The csv module simplifies working with CSV files by providing tools to read, write, and process tabular data.

cv2: the cv2 module, part of OpenCV, is a library for computer vision and image processing. It offers tools for tasks like image manipulation, object detection, and real-time video analysis. Common applications include robotics, face recognition, and augmented reality.

YOLO library from ULTralytics [11] [12]: in this project, we employ the Ultralytics YOLO Python library for object detection and inference tasks. This library turned out to be crucial for our project because it simplifies the process of handling model outputs through well-structured classes like Boxes (which allow users to access bounding box coordinates), confidence scores, and predicted classes directly.

5. Work Phases

5.1. Aria glasses exploration

Our work started with an exploration of the tools provided to us by the Aria Team, looking for all possible important information on the website. Then, we proceeded to install all the required software so we could start taking the first recordings. We paired the glasses with the mobile companion app and started the glasses calibration, then, we explored the applications' features, learning the possibility to create multiple profiles.

We took some videos moving some objects on a table. To view the recordings, we connected the glasses to a computer and used Aria Studio, which redirects to the Rerun program, to access the VRS files.

We then proceeded to request the MPS of interest (the eye gaze ones). Those were also visualized through the Rerun program alongside the VRS recording. Figure 9 shows an example of visualization of MPS eye gaze on the VRS with Rerun. From that, we gained a "visualization" of the outputs that helped us understanding what we could do in practice with those MPS points.

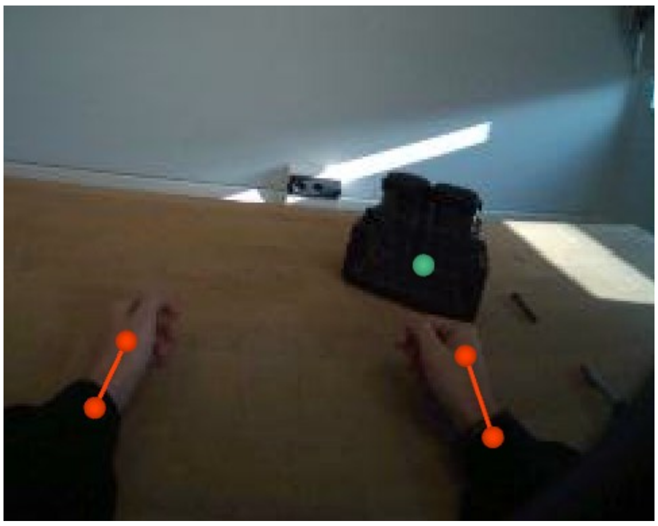


Figure 9: An example of visualization of a frame with Rerun

5.2. Pictures labelling

During our second visit to the Mechanical Department, we recorded some videos of the disassembly operation (the operation n.12) and took pictures of the brake piece, the screws, and the screwdriver used in the operation.

After that, it followed a selection of those pictures and the extraction of frames from a video recorded in the lab session. The extraction of frames from the MPS was carried out using a code ("Model Training\extract_each_n_frame.py") developed by us, which extracts one frame every n frames of a specified camera from a VRS format file, where n is a specified number of frames provided as input.

We included both extracted video frames and photos in the training and validation dataset. This decision ensures that the frames better represent the scenarios we aim to evaluate during the experimental phase.

We selected one hundred pictures and frames to shape our algorithm. To make the algorithm more robust with the detection of screws and screwdrivers, we decided to add, only to the training set, one hundred already-labeled pictures of screws and one hundred already-labeled pictures of screwdrivers from two datasets available online: Screws Dataset, Screwdriver Dataset.

The distribution of the dataset we decided to apply is:

- Training set: 270 pictures, composed by two hundred already labelled pictures (the one of the screws and screwdrivers) and 70 pictures and frames taken by us.
- Validation set: 18 pictures, composed by pictures and frames taken by us.
- Testing set: 12 pictures, composed by pictures and frames taken by us.

The labels we included in our algorithm are:

- Brake_back: with that label we identified all the fraction of the images in which all of the back of the brake was clearly visible.
- Brake_front: with that label we identified all the fraction of the images in which all of the front of the brake was clearly visible.
- Brake_general: with that label we identified all the brake appearances that could not be identified with the two previous labels.
- Screw.
- Torx_8940.

We then proceeded to map all the labels in all the selected pictures and frames. In the end, the count of boxes labelled was: 37 for Brake_back, 25 for Brake_front, 53 for Brake_general, 391 for Screw, and 133 for Torx_8940.

To make the training phase more robust, we decide to apply data augmentations to our training set.

To choose which type of data augmentations to apply to our training set, we performed two different sets of augmentations and then tested which one of the two models trained performed better in the testing phase.

The first set of augmentations applied were:

- 50% of probability of horizontal flip (mirror effect).
- Exposure: shuffling of exposure between -15% and +15%.
- Blur: up to 1px radius.

The second set of augmentations applied were:

- 50% of probability of horizontal flip (mirror effect).
- Brightness: shuffling of brightness from -25% to +25%.
- Noise: up to 1.8% randomly selected pixels were covered by a black point.

Each of the two augmentations sets was made by taking advantage of Roboflow's data augmentations functionality, which allows producing up to three versions of the same photo with some specific augmentations applied. Shuffling those three effects across the training set pictures, we tripled the number of pictures in both cases.

Finally, we downloaded the datasets with the labeling applied. The downloaded datasets are available in "Model training\Dataset_Exposure_Blur" and in "Model training\Dataset_Brightness_Noise".

In the next paragraph, we explain which types of augmentations we choose in our model and why.

5.3. Algorithm training

After the labelling phase, the datasets were saved in a folder named dataset accordingly to the standard compatible with the Yolo framework. The folder is structured as follows:

dataset

- data.yaml
- train

  - images
  - labels
- valid
  - images
  - labels
- test
  - images
  - labels

The *data.yaml* file is a text file that provides useful information to the algorithm, such as the number of classes and their names, while in the labels folder, the text files describe the box position and size for each picture.

Once the folder was ready, we proceeded to train the YOLO algorithm. The folder was uploaded to Google Drive, making it accessible via Google Colab. On Google Colab we could lever the available GPU to train, validate and test our model. Training, validation and testing were performed using the Ultralytics python library functions. The training phase uses a batch size of 16 and runs for 50 epochs.

The Python code we used is consultable in "Model Training\training.ipynb", while the trained models' weights are stored in "Model Training\Models weights (.pt files)".

To assess the quality of the model, we took into consideration the test phase metrics. The indexes selected for our comparison are:

- **Overall Precision**: the ratio of true positive detections to the sum of true positive and false positive detections across all classes.
- **Recall**: the ratio of true positive detections to the sum of true positive and false negative detections across all classes.
- **mAP@50**: The Mean Average Precision calculated with a minimum Intersection over Union (IoU) threshold of 50%. A detection is considered correct if the IoU with the ground truth is at least 50%.
- **mAP@50-95**: The Mean Average Precision computed over multiple IoU thresholds ranging from 50% to 95%, typically in 5% increments. This provides a comprehensive evaluation across different levels of overlap between detected and ground-truth objects.

- **mAP@50 for Each Class**: The Mean Average Precision at a 50% IoU threshold, calculated separately for each object class. For the comparison between the two configurations, the minimum value of this list of indexes is used as the representative score.
- **Fitness:** An overall metric for the goodness of the model.

The values of these metrics for two data augmentation configurations are shown below:

	Mirror effect+	
	Blur+Exposure	Noise+Brightness
Overall precision	0,536	0,583
Recall	0,554	0,807
mAP@50	0,578	0,664
mAP@50-95	0,472	0,572
min{mAP@50 for each class}	0,011	0,101
Fitness	0,483	0,581

In the end, the gap in Overall precision and in minimum mAP@50 for each class made us prefer the second set of augmentations.

Unfortunately, the minimum mAP yielded bad results in both cases. Additionally, most of the time screws placed on the table were identified as Torx. We identified two main justifications for this phenomenon:

- We used the same label both for screws on the table and for screws mounted in the piece.
- The screws we labelled were very small and had low resolution, especially the ones mounted in the component.

To address this issue, we took the following steps:

- The number of Torx labels was reduced from **133** to **99**.
- We divided the screw labels into two subcategories: **screws_in** (the screws in the brake component) and **screws_out**.
- We eliminated most of the screw's images coming from the web dataset leaving only the images where the screws were more similar in length and shape to the screws used in the process.
- We captured 12 new screws pictures, more like the ones used in the process, and we added them to the training set. Each picture contained multiple screws.
- We captured 13 new screws pictures directly from videos recorded with Aria Glasses.

- We performed data augmentation using a code developed by us, “Model Training\Data Augmentation\Augmentation_script.py” on the new screw’s pictures (in addition to the augmentation done with Roboflow) multiplying in this way the number of new pictures by 4 for a total of 100 new pictures. The *augmentation_script* reads images from an input folder, randomly applies various augmentations, and saves the results to an output folder. Key augmentations include brightness adjustment, color variation, rotation, resizing with padding, homography and gaussian noise.

The final dataset is stored in “Model Training\Dataset_Final_Model”. The final number of labels for each class is reported in the table below.

Brake_general	136
Brake_front	90
Brake_back	149
Screw_in	153
Screw_out	1608
Torx_8940	297

The test performance results in this case were the following:

Overall precision	0,617
Recall	0,736
mAP@50	0,688
mAP@50-95	0,474
min{mAP@50 for each class}	0,087
Fitness	0,495

Although the minimum mAP remained very low, we further analysed the mAP array:

Brake_back	Brake_front	Brake_general	Screw_in	Screw_out	Torx_8940
0,305	0,7084	0,812	0,087	0,692	0,238

Some results were slightly reduced due to these changes, but the recognition of **screws_out** improved significantly. The **screws_in** category continued to yield poor results however, we decided this was acceptable because **screws_in** are always contained within the bounding box of the part. Therefore, their identification does not affect the outcome.

Based on this analysis, we decided to proceed with this model.

5.4. Eye Gaze coordinates conversion

To convert the MPS eye gaze output, we developed the function *extract_eye_gaze_coordinates*, that, taking advantage of the Aria Glasses Python library, takes all the information needed to re-project the 3D coordinates of the eye gaze into the 2D frames and gives as output a list of tuples containing the coordinates of the eye gaze per frame and the associated timestamp. The code snippets used and adapted for this part of the code are available on the Aria Project website [7].

The process begins by reading the MPS file, which contains eye gaze angles and positions relative to the camera's frame of reference. These 3D data points are reprojected onto the 2D plane by assuming a default depth of one meter[2]. For each gaze entry, the script identifies the closest timestamp and retrieves the corresponding gaze information. Video data from the VRS file provides calibration details, ensuring accurate reprojection of the gaze vector onto the 2D image. The resulting 2D coordinates and timestamps are then stored as a list of tuples.

5.5. Inference on frames

To identify bounding boxes in images, we developed the function *boxes_from_image* to extract those boxes, convert them, group them in a convenient way, and store them in a list to return. This function takes an image as input and processes it using the trained YOLOv11 object recognition algorithm. While YOLOv11 outputs bounding box data, its format is not immediately usable for our purposes; so our function extracts bounding box coordinates, converts them into a standardized tensor format (xy), and then into a numpy list. This xy format specifies the rectangle's minimum and maximum x and y coordinates, uniquely defining the bounding box. The Ultralytics Python library functions [11][12] are used for this conversion. The final output is a list containing bounding box coordinates and associated classes, ready for further processing.

5.6. Testing of eye gaze inside bounding box

Two functions have been developed to determine whether a point, specifically, the eye gaze coordinate provided as input, falls within one of the predefined allowed bounding boxes. The concept of "allowed bounding boxes" is used as input to define what stage of the process we are in. By specifying these bounding boxes as input, the code can be executed with a flexible set of permitted objects tailored to the task at hand. For example, during a removal stage involving only hands, the brake is the sole allowed object. In contrast, during an unscrewing stage, screws and screwdrivers must also be included as allowed objects. Ultimately, these two

[2] This default distance was then modified after the experiment phase, in which is explained the motivation of this change.

functions return a Boolean value: True if the point lies within any of the allowed bounding boxes, and False otherwise.

The first function (*test_point_inside*) serves as a supplementary mechanism to verify whether gaze coordinates fall within a given single bounding box. This function ensures that the gaze analysis remains efficient, modifying the evaluation flag only if the gaze is within the bounding box.

The second (*test_frame*), examines the bounding boxes detected in the frame and cross-references them with the gaze coordinates. By filtering for allowed classes, the function determines if the gaze point lies within any of the bounding boxes. If a box or more are found, it flags the frames (one at a time) to the other functions, updating, if needed, the "flag" variable.

Finally, an overlay function (*put_overlay*) was developed to apply a green or red effect to frames based on the flag status, providing visual feedback on gaze-object interactions.

5.7. Video processing

The main function we developed, *inference_video*, integrates object detection visualization, eye gaze tracking, and flag visualization.

The function starts by reading video data and synchronizing it with eye gaze coordinates extracted from the *extract_eye_gaze_coordinates* function. Then we used code snippets available on the Aria Project documentation (that use the functions from Aria Project library) to read the VRS object. Bounding boxes for detected objects are obtained for every frame with the *boxes_from_image* function, and the gaze is cross-referenced with these boxes to determine if it aligns with predefined object classes (with the *test_frame* function). Depending on the outcome, visual feedback is added to the frame in the form of overlays (with the *put_overlay* function): green overlay indicates a successful gaze-object interaction, while a red one highlights a mismatch. Additionally, the system annotates frames with bounding box details and gaze points, enabling a clear visual representation of the analysis.

All the frames are then stored in a VideoWriter object (of the cv2 Python library), and this object is converted and saved as mp4 format for easy visualization.

5.8. Notebook "test_video"

All the previously described functions are stored and are consultable in a file called functions.py ("Results visualization\functions.py").

A notebook *test_video* ("Results visualization\test_video.ipynb") has been written with the purpose to store in it the paths to the file, the trained model weights, and to give those entities to the processing function, calling it from the functions.py file.

6. Experiment and results

6.1. Choice of confidence level for our algorithm

For this project, we set a threshold of 0.6 as the minimum confidence level required for the model to consider detection as valid. Setting its value equal to 0.6 ensures that the bounding boxes generated by the model are accurate and relevant. This helps in reducing false positives, such as detecting irrelevant objects or incorrectly identifying focus points. While this choice might result in some missed detections (false negatives), it provides a more precise foundation for understanding attention and focus patterns, making the system's outputs more reliable for real-world use.

6.2. Experiment

The video stored in "Experiment and results\files\VRS_def.vrs", was processed from frame 650 to 1120 (47 seconds), a time window that includes all relevant operations to calculate the Object Recognition index. This time window is selected because it contains all the necessary operations and actions relevant to our test process. We then proceeded to map, with a Python code ("Experiment and results\object_recognition_index.ipynb"), the interval of time in which a specific object is in the video, from frame to frame. Frames containing torx or screws overlapping with the brake were excluded: in our algorithm, if a bounding box is bounded inside another bounding box, then the first one is irrelevant in any case. As we can see in figure 11, the screw mounted in the brake is not detected by the algorithm. Nevertheless, if the operator is looking at the screw the result would be the same, because the eventual bounding box of the screw would be completely enclosed in the brake one.

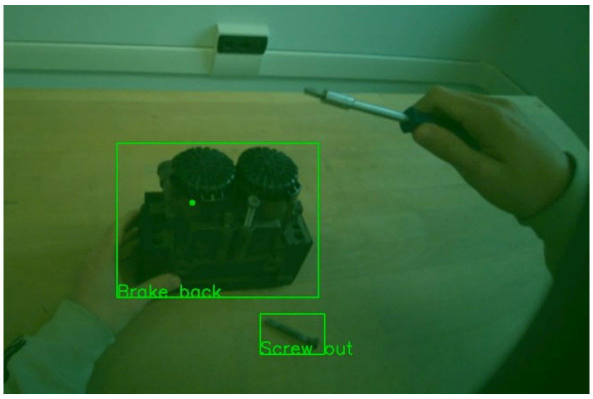


Figure 10: Example of screw mounted inside the brake

To test the Gaze Accuracy index instead, by personal inspection, we process the tester video with grids every 50 pixels with a Python code ("Experiment and results\eye_gaze_index.ipynb") and determine the precision of the eye gaze by confronting it with the point that the glasses wearer was actually looking at.

6.3. Results

The eye gaze testing performed well overall, although there was a slight shift to the left in the estimated gaze point compared to the actual gaze location. This displacement is likely due to the fact that the MPS processing is done in a 3D coordinate system, while we convert the data into a 2D coordinate system, assuming a fixed distance of 1 meter. Based on video analysis, we estimate the displacement to be around 60 pixels (slightly more than 50 pixels).
After the computation:

- the Object Recognition Accuracy index turned out to be 75.82%
- the Gaze Accuracy index turned out to be: $1 - \frac{60}{1408} = 95.74\%$

We further investigated the Gaze displacement of 60 pixels, trying to find what went wrong in the conversion. We found that, diminishing the default eye gaze depth for the conversion, the left displacement diminishes as well. So, we estimated the distance between the operator and the workspace to be 0.4 meter (the distance in which there is no displacement), and we corrected the code accordingly.
The new Gaze Accuracy, since there is no displacement left after the change in the default depth, is equal to 1.
We can then compute our Distraction Detection index

$$Distraction\ detection\ accuracy = Eye\ Gaze\ Accuracy * Object\ Recognition\ Accuracy = \mathbf{75.82\%}$$

6.4. Example of Proof of Concept

The video saved in "Results visualization\video_flag.mp4" is the experiment final result. In this video, the 12th operation is performed by the operator and the algorithm print a True if the operator is focused on the operation's objects and False otherwise.

7. Future developments

The results of our work were satisfactory but not optimal. In particular, the object detection algorithm was not enough accurate in detecting the screws mounted in the brake component. Furthermore, the algorithm struggled with the correct identification of the main component orientation. To overcome the orientation problem, we tried to improve the dataset increasing the number of the brake pictures and balancing the number of Brake_front, Brake_back and Brake general labels. However, we didn't achieve the expected improvement (additional info about this attempt can be found in the Annex 1). Regarding the torx and the screws on the table detection, the algorithm achieved good results. Also the projection of the wearer eye gaze and the global logic of the algorithm worked properly detecting correctly whether the operator was looking to the objects or not.

Future developments must be directed to the training of a Hidden Markov Model for the process stage recognition. This work includes the definition of the model's emissions, the data collection about the disassembly process, and the model training. To test and finally use the HMM, it is necessary to develop the code to compute the emissions value from the Aria Glasses cameras and sensors. To improve the performance of the stage recognition algorithm, the use of external cameras and sensors may be evaluated to increase the amount of available data.

An additional area for improvement could be real-time data processing. Currently, this is limited by the MPS constraint, which prevents real-time data processing, and the Aria Glasses connectivity, which restricts real-time data exchange.

8. Conclusions

This project explores the use of Project Aria Glasses for distraction detection in manufacturing environments, integrating neural network-based object detection and gaze tracking. The developed proof of concept demonstrates the feasibility of using ego-centric monitoring systems to identify operator distractions during task execution. By leveraging innovative hardware, such as the Aria Glasses, and advanced algorithms like YOLOv11, the project effectively showcases the potential for enhanced productivity and safety in industrial settings.

Key lessons learned include the importance of robust training datasets for accurate object detection and the need to mitigate limitations introduced by hardware and connectivity constraints. The system's performance is satisfactory, with strong results in gaze tracking and bounding box identification. However, the misclassification of the component orientation and screws highlights areas where the object detection model could benefit from improved dataset diversity and higher-resolution training images.

The authors contribute to this research by designing and implementing a proof of concept that integrates gaze tracking, object detection, and task monitoring to address distraction detection in industrial settings. Key contributions include the development of a pipeline for processing Aria Glasses data, training a YOLOv11 model for object detection, and proposing a distraction detection framework. The team also identifies practical limitations, such as hardware constraints and dataset quality issues, and provides actionable recommendations for future improvements.

Annex 1

From the experiment video emerged one critical issue. When the operator get closer to the table and the perspective is no more perpendicular to the piece's frontal face, the detection algorithm misclassifies the front face as the back face. Given this result we made a final attempt to fix the problem. We increased the number of brake labels and tried to balance the number of brake_front, brake_back and brake_general labels. All the new pictures were taken from videos recorded with the Aria glasses. We added 8 new pictures that then we augmented using the "Model Training\Data Augmentation\Augmentation_script.py" obtaining 40 new pictures that were further incremented on the Roboflow platform reaching at the end 120 new pictures. The dataset of this attempt is available in "Model Training\Dataset_Last_Attempt.py".
The final number of labels for each class was the following:

Brake_general	159
Brake_front	150
Brake_back	156
Screw_in	153
Screw_out	1608
Torx_8940	297

Unfortunately, the results were not the expected ones. The algorithm started again to confuse the screw placed on the table as Torx. At this point we tried to increase the number of epochs from 50 to 75 and the batch size from 16 to 32 but it led to no improvements.
We couldn't find an explanation for this and so we decided to proceed with the object detection algorithm trained before.

References

[1]https://www.emerald.com/insight/content/doi/10.1108/01443579710182936/full/html?

[2] https://www.inpixon.com/blog/cost-of-digital-distraction-in-industrial-environments

[3] https://facebookresearch.github.io/projectaria_tools/docs/intro

[4] aws.amazon.com

[5] https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network

[6] https://www.nature.com/articles/s41598-023-49739-1#Sec22

[7] https://www.projectaria.com/

[8] https://www.v7labs.com/blog/yolo-object-detection

[9] https://docs.ultralytics.com/

[10] https://roboflow.com/

[11] https://docs.ultralytics.com/modes/predict/#images

[12] https://docs.ultralytics.com/reference/engine/results/#ultralytics.engine.results.Boxes