FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Eye Tracking User Interface

Luís Tiago Galvão Ferreira

U. PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Prof. António Miguel Pontes Pimenta Monteiro

July 25, 2020

Eye Tracking User Interface

Luís Tiago Galvão Ferreira

Mestrado Integrado em Engenharia Informática e Computação

July 25, 2020

Abstract

The Industry 4.0 paradigm demands a constant improvement of manufacturing processes. We can quickly imagine artificial intelligence putting machines doing things to help the daily human tasks.

Thinking on the shop-floor, it's common to see operators wearing gloves, maneuvering heavy or sensitive material, having their hands occupied. Sometimes, these operators need to interact with the software systems used in manufacturing, to track and document the transformation of raw materials to finished goods, without using their hands.

The ideal solution is the use of a strategy that integrates specialized hardware to track the human gaze in real-time with manufacturing systems.

Tracking eyes has been a subject of research and development in many fields. There are many hardware devices for *Eye Tracking*, some of them are used over the head and have a high degree of accuracy, while others use structured infrared lighting and pré-calibration to track where the user is looking on the screen.

In this project, it will be explored not only the different *Eye Tracking* algorithms and hardware devices but also alternatives for *GUI* interactions, like *Hand Tracking* and *Voice Recognition*, to provide operators with tools that can help them in their work.

Using *Machine Learning* algorithms, proved to be an important resource for prediction and detection of the eye or hand movements, triggering a respective action in the *GUI*.

The movements with the eyes, using *Linear SVM* algorithm for prediction, was proved to be a great feature, not only for the quick response of 3,37 seconds on average and for their accuracy, but also for being an asset when it comes to using extra validations to perform an action in the manufacturing software systems.

Regarding the movements with the hands, we conclude that the time of model training is not the factor that will increase the accuracy, but the size of the training dataset. The *Averaged Perceptron* algorithm, associated with the hand movements, takes 14,48 seconds since the user performs the action until the algorithm returns the result, with 92,86 % of accuracy.

Finally, it was concluded that the user can interact with manufacturing software systems without using a mouse or keyboard, and customize this interaction in any way he wants, making the work more efficient.

Keywords: Software, Computer Vision, Eye Tracking, Hand Tracking, Machine Learning, Manufacturing, Industry 4.0

Resumo

O paradigma da Indústria 4.0 exige uma melhoria constante dos processos industriais. Rapidamente imaginamos a inteligência artificial a colocar máquinas a realizar operações com o intuito de facilitar as tarefas dos humanos no seu dia-a-dia.

Pensando no chão de fábrica, é comum os operadores utilizarem luvas para pegar em material pesado ou sensível, com as mãos ocupadas. Às vezes, esses operadores precisam de interagir com sistemas de software industriais sem mãos, de modo a documentarem e validarem o processo associado à transformação de matérias-primas em produtos acabados.

A solução ideal para este problema, consiste na utilização de uma estratégia que interligue um hardware com múltiplas câmaras, de modo a monitorizar o olhar humano, em tempo real, de forma a ser possível executar operações em sistemas de software industriais.

Monitorizar os olhos tem sido objeto de pesquisa e desenvolvimento em muitos campos. Existem muitos dispositivos de hardware para monitorização ocular. Alguns são montados diretamente na cabeça e têm um alto nível de precisão, enquanto outros utilizam iluminação infravermelha estruturada e pré-calibração, para calcularem para onde o utilizador está a olhar.

Neste projeto serão investigados não só os vários algoritmos e hardware associado à monitorização ocular, mas também outras alternativas para a interação com a *GUI*. Como por exemplo a deteção de mãos e respetivos movimentos, e reconhecimento de voz. De maneira a fornecer aos operadores ferramentas que os possam ajudar no seu trabalho.

O uso de algoritmos de *Machine Learning*, provou ser um importante recurso no que diz respeito a previsão e deteção de movimentos com os olhos ou mãos, desencadeando uma ação na *GUI*.

Os movimentos com os olhos, utilizando o algoritmo *Linear SVM* , provou ser uma grande funcionalidade não só pela resposta rápida de 3,37 segundos em média e pela sua precisão, mas também por ser uma mais valia no que diz respeito ao uso de validações extras para executar uma ação nos sistemas de software industriais.

Em relação aos movimentos com as mãos, concluímos que o tempo de treino do modelo não é o fator que aumenta a precisão, mas sim o tamanho do conjunto de dados de treino. O algoritmo *Averaged Perceptron*, associado aos movimentos das mãos, demora cerca de 14,48 segundos desde que o utilizador executa a ação até que o algoritmo devolve o resultado, com 92,86 % de precisão.

Para finalzar, foi concluido que o utilizador pode interagir com sistemas de software industriais sem o auxílio de rato ou teclado, e personalizar essa interação da maneira que desejar, tornando assim o trabalho mais eficiente.

Keywords: Software, Visão por Computador, Monitorização ocular, Monitorização gestual, Indústria 4.0

Acknowledgements

I would like to thank everyone involved in this project, it was a period of hard work.

To Prof. Miguel Pimenta Monteiro, for the support and guidance provided.

To José Pedro Silva, for the opportunity he gave me to contribute with this project to Critical Manufacturing Product and for sharing his knowledge and his ideas.

To João Gonçalves for sharing his knowledge and giving me good advices. Finally, I thank to my family, girlfriend, and friends who never stopped believing in my capabilities, and for unconditional emotional support.

Tiago Galvão

"Innovation distinguishes between a leader and a follower."

Steve Jobs

Contents

List of Figures

List of Tables

Abbreviations

APPD	Adaptive and Precise Pupil Boundary Detection
CHT	Circular Hough Transform
CIM	Computer Integrated Manufacturing
CMF	Critical Manufacturing
CNN	Convolutional Neural Networks
EAR	Eye aspect ratio
ElSe	Ellipse Selection
ExCuSe	Exclusive Curve Selector
GUI	Graphical User Interface
IIS	Internet Information Services
MES	Manufacturing Execution System
MESA	Manufacturing Enterprise Solutions Association
ML	Machine learning
MRP	Material Requirements Planning
MRP II	Manufacturing Resources Planning
PCCR	Pupil centre corneal reflection
SET	Sinusoidal Eye-Tracker
STT	Speech-To-Text
UX	User Experience

Chapter 1

Introduction

1.1 Context

Manufacturing execution systems (MES) are computerized systems used in manufacturing, to track and document the transformation of raw materials to finished goods. These systems provides information that helps manufacturing decision makers understand how current conditions on the plant floor can be optimized to improve production output.

The challenges created by globalized manufacturing business used *MES* as pivotal in the performance, quality and agility. However, a completely new generation is required to the new challenges created by Industry 4.0. [3]

1.2 Motivation

The Industry 4.0 paradigm demands a constantly improvement of manufacturing processes. We see more and more tasks becoming automated, an increase in online interaction with the process and an overall shift towards cyber physical systems.

Thinking on the shop-floor, it's common to see operators wearing gloves, maneuvering heavy or sensitive material, performing complex manufacturing processes while interacting with computer systems, having their hands occupied.

Performing several tasks at the same time it's not easy, and often leads to unwanted failures. If we are working with expensive and fragile components it's recommended that we have a lot of attention, an example is the *Wafers* manufacturing case, characterized next.

In electronics, a wafer is a thin slice of semiconductor used for the production of integrated circuits or to manufacture solar cells in case of equipments. The production process for this component involves many micro fabrication processes, such as doping, ion implantation, etching, thin-film deposition of various materials, and photolithographic patterning. The individual microcircuits are separated by wafer dicing and packaged as an integrated circuit [19].

So it's very useful that manufacturing operators can interact with software manufacturing systems without using their hands in a quick and simple way. Simple tasks like assemble and disassemble components while seeing information about the next steps of this process in a computer, access the statics of the some process, or even documenting something, becomes easier.

1.3 Objectives

The main goal addressed in this work is to integrate the *MES Critical Manufacturing System* with an *Eye Tracking* platform so the operators can interact with the *MES* without the need to use their hands.

The usage of an *Eye Tracking* solution has several benefits towards Industry 4.0. It can be used to help with skills transfer and training, or to streamline processes and identify waste or roadblocks in a system, and it can also be used to improve safety and increase situational awareness.[4]

Eye Tracking allows you to see roles and tasks from the perspective of the worker. It gives clear and accurate data on where they are looking and how they perform tasks, and this information reveals a lot about subconscious processes and behaviors which workers may be unaware of or unable to articulate.[4]

1.4 Document Structure

This document is divided into a set of chapters that aims to investigate and propose a solution for interaction with *MES* systems totally hands-free.

Starting with chapter 1, it will be presented a brief description of *MES* systems, followed by *Eye Tracking* historical position, as well as main algorithms used nowadays. *Machine Learning* applied to *Computer Vision* will also be a point that will be discussed. Chapter 2 will be concluded with related work, the applications and the experiences made in *Intelligent Systems, Interaction and Multimedia Seminar* on 2019.

In chapter 3, the proposed solution is presented and also a set of use cases associated with functional and non-functional requirements.

Chapter 4 contains the description of the solution architecture, as well as a brief description of the architecture of the *Critical Manufacturing* product, where this project will be applied.

In chapter 5, some important details of the implementation are presented and the found solutions analysed.

This document concludes with chapter 6, describes some analysis, testing and experiments related with this project, and chapter 7 where main contributions and future work are presented.

Chapter 2

Eye Tracking Techniques

In this chapter a brief description of *MES* systems is presented, followed by *Eye Tracking* historical position, as well as main algorithms used nowadays. *Machine Learning* applied to *Computer Vision* is also a point mentioned and discussed. It concludes with related work, the applications and the experiences made in *Intelligent Systems, Interaction and Multimedia Seminar* on 2019.

2.1 Manufacturing Execution Systems

Competition has grown exponentially in recent decades, leading companies to take several measures to increase productivity and reduce costs. This combination has been implemented in several ways, ranging from the transfer of production units to countries with low labor costs, to the use of industrial management software and other computerized tools to plan, control and increase the productive performance of human resources and materials [36].

The use of software systems to support production activities began with the rise of large computers, created in the 1940s, being only incorporated in conventional companies in the 1950s and 1960s. They were implemented to support the financial departments, later applied to production departments, taking in first place analysis of costs and inventory, then the planning and control [36].

The support to the industry has evolved and caused the arising of several types of systems, such as *MRP - Material Requirements Planning*, in the 60s, which converts a production plan into a list of the quantities of materials needed for its execution, and *MRP II - Manufacturing Resources Planning*, in the 1980s, which expands the range of analysis of the previous tool, incorporating needs for human resources, equipment, energy, financial resources. In this period these solutions were generically treated as *CIM - Computer Integrated Manufacturing* [36].

As an evolution of the *MRP II* concept, the integrated *Enterprise Resource Planning* solutions emerged, which end up being an eminently administrative-financial tool, giving a little support to production, especially through the logistics and production sectors [36].

One of the most recent elements of this set is the *MES - Manufacturing Execution System*, which allows the planning, control and monitoring in real time of the various processes and production stages, both in terms of physical quantities of raw materials, intermediate and finished products, as well as in terms of financial costs of any kind - personnel, energy, raw materials, intermediate products. *MES* performs all the functions of *MRP* and *MRP II* and makes a bridge between planning and production control, directly from equipment such as scales and barcodes, or manually, but always in real time [36].

The name *MES* was created by the company *AMR Research* in 1990 and in 1992 a group of software development consultants and solution integrators created the *MESA - Manufacturing Enterprise Solutions Association*, which defined a set of eleven features that would be mandatory for any application proposed to obtain this classification and that is known as *MESA* model [36].

In 2000, the *International Society of Automation* developed the first part of the *ANSI-ISA-95* standard and three more until the year 2010. This standard establishes a set of terminologies, activities and functions, as well as the hierarchy of these functions, and a standardization between the interfaces of *MES* system [36].

Critical Manufacturing MES provides manufacturers in demanding discrete industries a platform for Industry 4.0 success. It delivers reliable access to detailed and timely operational information with full context and intelligence for fast, confident decisions and profitable action [23].

Whether plants use traditional technologies or Industry 4.0 distributed intelligence, the *Critical Manufacturing MES* is ready to foster progress and improvement. Users can readily configure, distribute and use workflows and screens [23].

Inherently designed to accommodate the *Industrial Internet of Things (IIoT)*, mobile devices, automation, sensors, *Critical Manufacturing MES* is truly an Industry 4.0 hub. This *Augmented MES* offers not only includes advanced analytics and continuous improvement tools, but also a manufacturing digital twin, and quick, intuitive looks at performance [23].

2.2 Eye Tracking History

The first who noticed that we have something special in our eyes was *Louis Emile Javal*, french ophthalmology investigator. In 1879 he said that our eyes don't move continuously along a line of text, but make short rapid movements intermingled with short stops.[21]

Later, *Charles H. Judd* developed the eye movement camera, a non-intrusive *Eye Tracking* device that recorded motions of eyes on film allowing detailed study of eye motion.[21]

In 1931, *Earl James* and *Carl Taylor*, created the Ophthalmograph and Metronoscope, devices used to record the movement of eyes while reading and tools that trained people to read more effectively. They understood, as *Louis Javal* said, that *reading was not simply a smooth motion over words*. Instead, a reader scans several words, pauses a moment to comprehend them, and then scans again. The Ophthalmograph was used to measure readers hops and fixation.[21]

In the late 1990s, *EURO RSCG* began using *Eye Tracking* technology to measure and study reactions to information on the internet.[21]

In 2006, *Bunnyfoot* company, using *Eye Tracking* and physiological data, build a study that examined how effective advertising was in video games in virtual worlds with digital billboards. [21]

However the study of *Eye Tracking* wasn't accessible to everyone, because the hardware with a significant precision was very expensive. For many years *Eye Tracking* was used as a tool in education research and by medical researchers and physicians.

Nowadays we have the example of *Tobii Eye Tracker 4C* hardware (that I will use as a case of study in the next sections) which is more focused on the gaming world and simultaneously tracks the eyes and head.

2.3 Eye Tracking Algorithms

I will start with the question: *Why do we move our eyes?*

Eye movements have 3 main functions which are considered important when we process visual information:

- Place the information that interests us on the fovea (saccades, vestibular ocular reflex and smooth pursuit);
- Help bring objects into focus (vergence);
- Prevent stationary objects from fading perceptually (microsaccades, tremor and drift)

The human visual field is about 134 x 220 degrees and is divided into three main regions : foveal, parafoveal and peripheral.

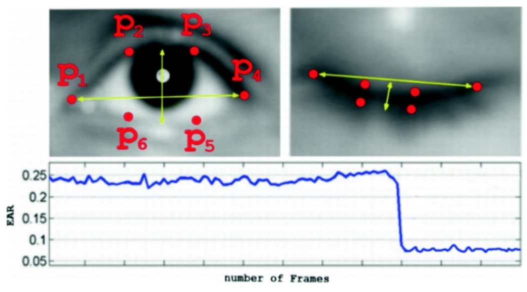
Our visual data is primarily registed in the foveal region, that represents less than 1% of the visual field. Even though this represents only a small part of our field of vision, the information registered through the foveal region constitutes about 10% of what is sent to the brain through our optic nerve [12].

The peripheral vision has very poor acuity, and main capabilities are for detecting contrasts and movements. So when we move our eyes to focus on a specific region of an object, we are placing the foveal region of the eye on top of the area that is currently within the main focus of the lens in our eye. This means that we are maximizing our visual processing resources on the particular area of the visual field that also has the best image due to the optic characteristics of the eye. By letting the foveal region register the image, the brain gets the highest possible image resolution of the interesting area to process as well as the most amount of data registered by the eye about that area [32].

Throughout the evolution of eye branch research, new ideas and discoveries emerged that led to the creation of universal eye detection algorithms. In the following subsections, some of the most relevant algorithms for *Eye Tracking* are examined.

2.3.1 Eye aspect ratio (EAR)

The concept of eye aspect ration (*EAR*) was introduced by *Tereza Soukupová and Jan Cech* in their 2016 paper, *Real-Time Eye Blink Detection Using Facial Landmarks* [37]. The *EAR* (Fig. 2.1.) involves a very simple calculation based on the ratio of distances between facial landmarks of the eyes. The *EAR* is mostly constant when the eye is open and is getting close to zero while it closes.



$$EAR = \frac{||p_2 - p_6|| + ||p_3 - p_5||}{2||p_1 - p_4||}$$

Figure 2.1: *EAR*

2.3.2 Circular Hough Transform (CHT)

Hough transform is one of basic methods which distinguish geometry from the image in an image processing. The circle candidates are produced by voting in the Hough parameter space and then selecting local maxima in an accumulator matrix.

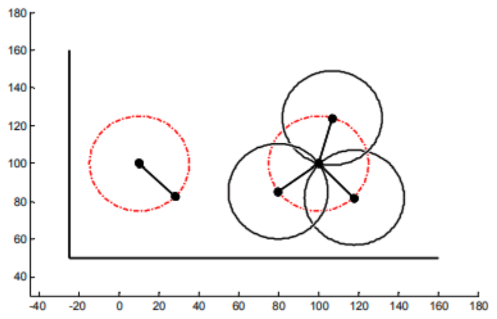


Figure 2.2: The principle of the *CHT*

This algorithm can be used to detect the pupil center. It uses x_c, x_y as coordinates of center and r as the radius of a circle:

$$\begin{aligned} x &= x_c + r * cos(\theta) \\ y &= y_c + r * sin(\theta) \end{aligned} \tag{2.1}$$

This method relies on the property of the edge points to describe circles with optimal radius r that intersect in the same point that is the center of the circle we want to detect [1].

2.3.3 Starburst algorithm

The *Starburst* algorithm was introduced by *Dongheng Li, Jason S Babcock, Derrick J. Parkhurst*, combining feature-based and model-based image processing approaches. The goal of the algorithm is to extract the locations of the pupil center and the corneal reflection so as to relate the vector difference between these location to coordinates in the scene image.[11]

Starburst uses the following steps:

1. Noise reduction
   Reducing the shot noise and line noise in the eye image applying a *5x5 Gaussian* filter with a standard deviation of 2 pixels.
2. Corneal reflection detection, localization and removal
   When infrared light is projected in the eye, the corneal reflection corresponds to one of the brightest regions in the eye image. It can be used an adaptive thresholding technique in each frame to localize the corneal reflection. The corneal reflection is removed by radial intensity interpolation, meaning that for each pixel between the center and the contour, the pixel intensity is determined via linear interpolation.
3. Pupil contour detection
   We start to detect edges along a limited number of rays that extend from a central best guess of the pupil center. For robustness to inaccuracy of the starting point, edges are also detected along a limited number of rays extending from the initial set of detected features returning in the direction of the starting point.
   For each frame, a location is chosen that represents the best guess of the pupil center in the frame.
   Next, the derivatives Δ along N, extending radially away from this starting point, are independently evaluated pixel by pixel until a threshold is exceeded.
   When this threshold is exceeded, a feature point is defined at that location and the processing along the ray is halted.
   The two-stage feature-detection process improves the robustness of the method to poor initial guesses for the starting point.
   At this point, an ellipse could be fitted to the candidate points, however, the bias would induce a significant error into the fit. To eliminate this bias, the two-stage feature-detection process is iterated [10].
4. Ellipse fitting
   Given a set of candidate feature points, we find the best fitting ellipse, using the *Random Sample Consensus (RANSAC)*, witch is an iterative procedure that selects many small but random subsets of the feature points, and uses each subset to fit an ellipse, in order to find the ellipse that has the largest agreement with the entire set of candidate feature points.

5. Calibration
   This step is related to the calculation of user's gaze point in the image scene, using a mapping between locations in the image scene and an eye position. During calibration, the user must look to a set of scene points for which the positions in the scene image are known.

2.3.4 Sinusoidal Eye-Tracker (SET)

The sinusoidal *Eye Tracker* is based on the deconstruction of contours in black and white images into sinusoidal components. The key feature of *SET* is the proper combination of manual and automatic steps that achieves high precision with a reasonable speed. Before applying the pupil detection we adjust the following parameters:

- The threshold for conversion of the eye image to a black and white image;
- The size of the segments considered for pupil detection.

This method is designed for *Eye Tracking* in a sequence of images. It uses the detected pupil position in the previous frame as the starting point of the search in the current frame. In cases where there is no prior pupil position, the center of the image is considered as the starting point.[17]

2.3.5 Swirski

Swirski is a real-time dark-pupil tracking algorithm designed for low-cost head-mounted active-IR hardware. This algorithm is robust to highly eccentric pupil ellipses and partial obstructions from eyelashes, making it suitable for use with cameras mounted close to the eye.

Swirski approach works in three stages:

- Approximate the pupil region using a fast, simple feature detection, to reduce the search space in the following stages;
- Use a k-means histogram segmentation to refine the approximation of the pupil region, and find an initial approximation to the pupil centre;
- Refine the pupil centre and find its elliptical outline, using a novel ellipse fitting algorithm [20].

2.3.6 Exclusive Curve Selector (ExCuSe)

ExCuSe is based on oriented histograms calculated via *Angular Integral Projection Function* and as input uses 8-bit gray-scale images. The coarse pupil center estimation is then refined by ellipse estimation similar to *Starburst*. This algorithm is evaluated on the *Swirski* dataset as well as nine other datasets that were collected during an on-road driving experiment and eight datasets that were collected during a supermarket study. The evaluation dataset consists of overall 38,401 images, where the pupil position was labeled manually on each image [39].

The workflow of the algorithm is shown in figure 2.3.

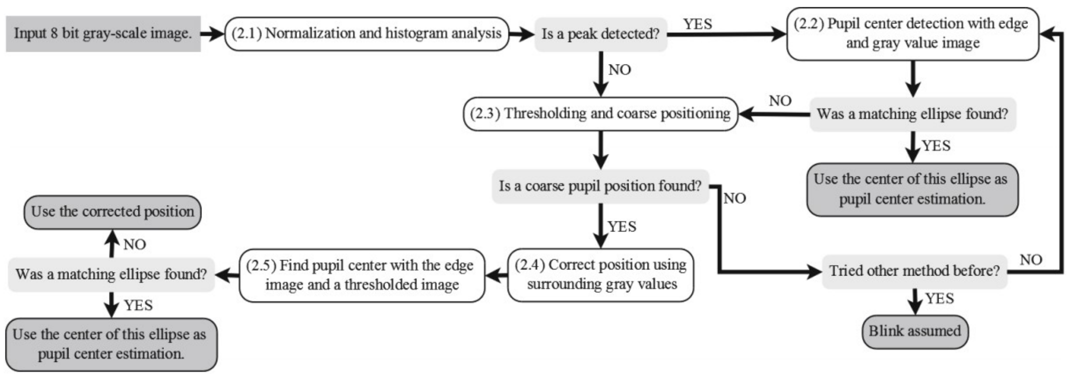

Input 8 bit gray-scale image.
(2.1) Normalization and histogram analysis
Is a peak detected?
YES
NO
(2.2) Pupil center detection with edge and gray value image
Was a matching ellipse found?
NO
YES
Use the center of this ellipse as pupil center estimation.
(2.3) Thresholding and coarse positioning
Is a coarse pupil position found?
NO
YES
Tried other method before?
NO
YES
Blink assumed
(2.4) Correct position using surrounding gray values
(2.5) Find pupil center with the edge image and a thresholded image
Was a matching ellipse found?
NO
Use the corrected position
YES
Use the center of this ellipse as pupil center estimation.

Figure 2.3: *ExCuSe* workflow

2.3.7 Ellipse Selection (ElSe)

ElSe proved high detection rates, robustness, and a faster runtime in comparison to *ExCuSe, SET, Starburst*, and *Swirski*.

ElSe operates on gray scale images, starting by calculate the edge image. Then we remove the edge connections that could impair the surrounding edge of the pupil.

Afterwards, the connected edges are collected and evaluated based on straightness, inner intensity value, elliptic properties, the possibility to fit an ellipse to it, and a pupil plausibility check. If a valid ellipse describing the pupil is found, it is returned as the result. In case no ellipse is found, a second analysis is conducted. To speed up the convolution with the surface difference and mean filter, the image is downscaled.

After applying the surface difference and mean filter to the rescaled image, the best position is selected by multiplying the result of both filters and selecting the maximum position. Choosing a pixel position in the downscaled image leads to a distance error of the pupil center in the full scale image. Therefore, the position has to be optimized on the full scale image based on an analysis of the surrounding pixels of the chosen position.[40]

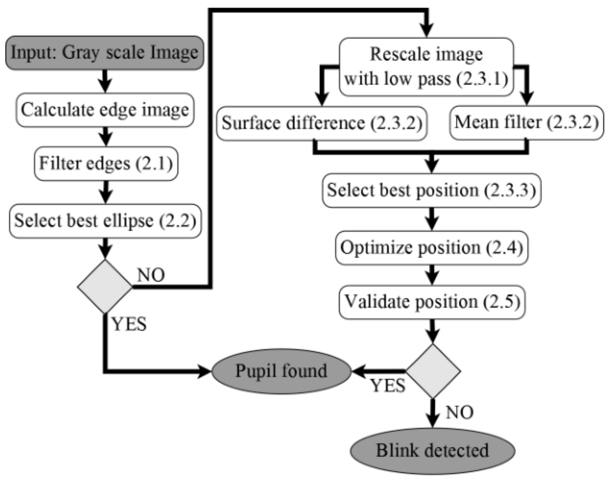


Figure 2.4: *ElSe* - algorithm steps

2.3.8 Adaptive and Precise Pupil Boundary Detection(APPD)

APPD follows a simple workflow and consists of the steps shown in figure 2.5, culminating in the contour detection of the pupil.

This adaptive method for pupil boundary infers if the pupil is severely occluded and spends more effort to detect the pupil without compromising real-time applicability.

The main strategy which improves the algorithm against occlusions is extracting the elliptical arcs from input image and finding one arc or a group of arcs representing the pupil contour. In this way, relevant features from a partially visible pupil can be extracted and detection can be performed by fusion of separate features. Besides detecting the pupil boundary and center precisely, the algorithm can also identify if there is no pupil in the image.[8]

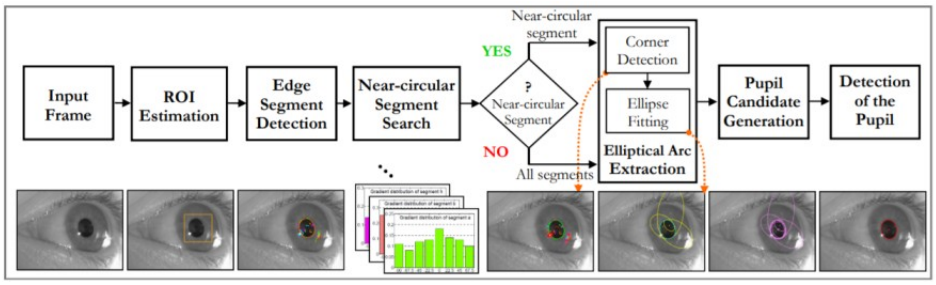


Figure 2.5: *APPD* Workflow

2.4 Machine Learning Applied to Computer Vision

Machine Learning (*ML*) is an important subset of artificial intelligence, for the scientific study of statistical models and algorithms which focuses on making predictions, without human explicit instructions.

ML algorithms build a mathematical model based on sample data, known as *training data*, in order to make predictions or decisions without being explicitly programmed to perform the task [18][6]. Those algorithms can be used in the computer vision field, for detecting eyes or hands movements, more easily than conventional algorithms.

There are some variations of how to define the types of *ML* algorithms but commonly they can be divided into categories according to their purpose where the main categories are the following [13]:

- **Unsupervised Learning**

  Unsupervised Learning describes a class of problems that involves using a model to describe or extract relationships in data.

  In *Unsupervised Learning*, there is no instructor or teacher, and the algorithm must learn to make sense of the data without this guide [15].

The goal in such *Unsupervised Learning* problems may be to discover groups of similar examples within the data, where it is called clustering, or to determine the distribution of data within the input space, known as density estimation, or to project the data from a high-dimensional space down to two or three dimensions for the purpose of visualization [35].

- **Reinforcement Learning**

  Reinforcement Learning describes a class of problems where an agent operates in an environment and must learn to operate using feedback.

  Reinforcement Learning is learning what to do — how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them [33].

- **Supervised Learning**

  The primary purpose of *Supervised Learning* is to scale the scope of data and to make predictions of unavailable, future or unseen data based on labeled sample data.

  It infers a function from labeled training data consisting of a set of training examples [28]. The function tries to model relationships and dependencies between the target prediction output and the input features, such that we can predict the output values for new data, based on those relationships which it learned from the previous datasets [14].

2.5 Previous Work

In the *Intelligent Systems, Interaction and Multimedia Seminar* on 2019, it was decided to explore the *Eye Tracking* world, developing two applications. This section will talk about some of this experiences.

The first application (*APP1*), made in python, used the traditional webcam associated to *OpenCV*, *dlib* and *imutils* libraries to detect the position of the eyes, and used the *EAR* algorithm to detect the eye blinking.

The second application (*APP2*), is a game about Highway Code, that can be played using *Tobii Eye Tracker 4C* hardware or a mouse, developed in C# and using the *Tobii Core SDK*.

Tracking the human eyes is a bit tricky, especially if it's done with a traditional webcam. The quality of light and image, are variables with a lot of weight. If we are in a dimly lit environment or too far from the camera or we rotate our head, it is practically impossible to detect the eyes (figure 2.6). Another limitation of the *EAR* algorithm is when we look down, confusing the resulting image to a closing eye.

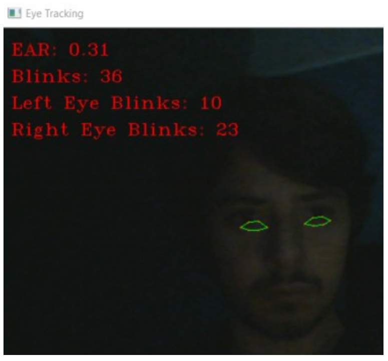


Figure 2.6: *APP1* - Open Eyes (bad light conditions)

If we are in an environment with moderate light, the *EAR* algorithm is applied correctly, and we have the eye blink detection for right, left and both eyes (figure 2.7).

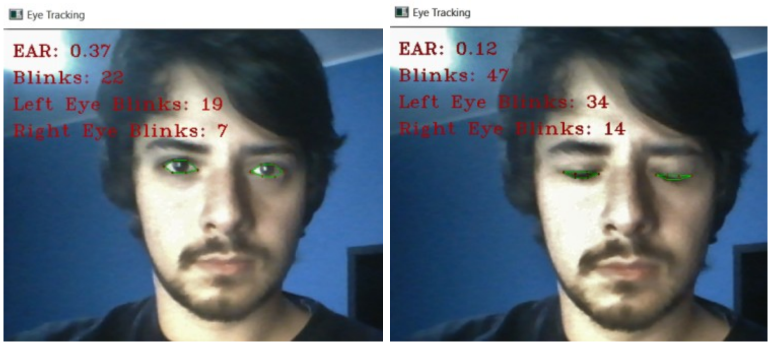
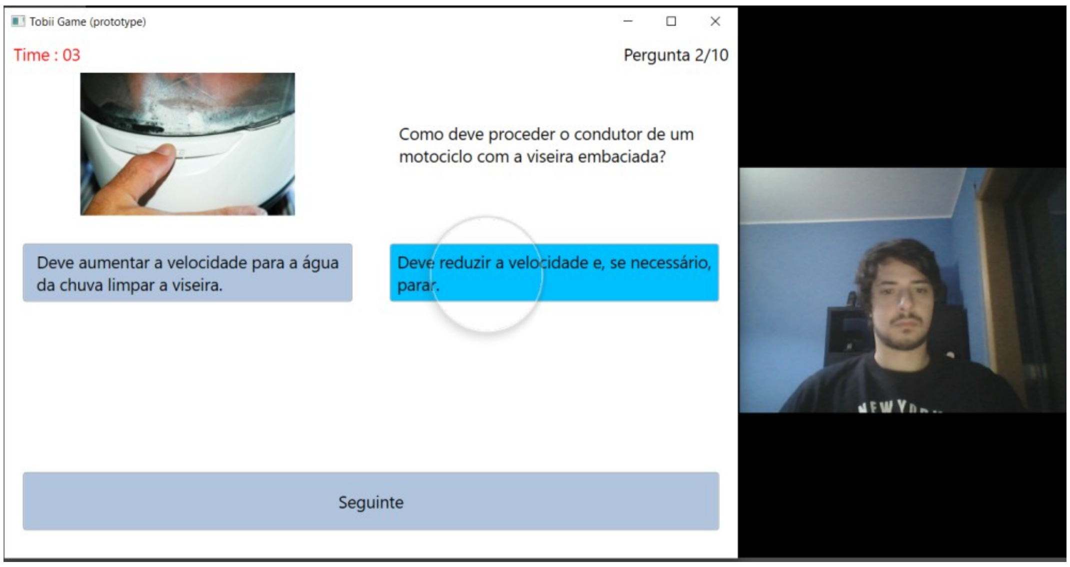


Figure 2.7: *APP1* - Open/Close Eyes (with good artificial light)

The *APP2* (figure 2.8) using *Tobii Eye Tracker 4C* or mouse was tested in 10 females and males from 15-75 years.



Como deve proceder o condutor de um motociclo com a viseira embaciada?

Deve aumentar a velocidade para a água da chuva limpar a viseira.

Deve reduzir a velocidade e, se necessário, parar.

Figure 2.8: *APP2* - Highway Code Game

For all individuals that don't know of the existence of this *Eye Tracking* platform, there are an initial adaptation phase. In the figure 2.9 we see that the *Eye Tracking* device can improve not only the user experience but the learning of the game itself, as the number of usage of the *APP2* increases.

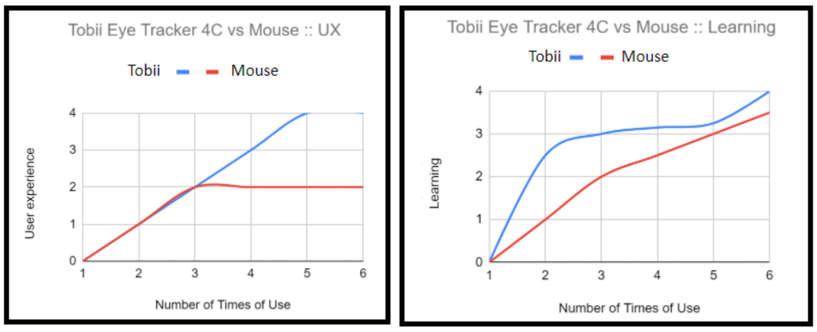
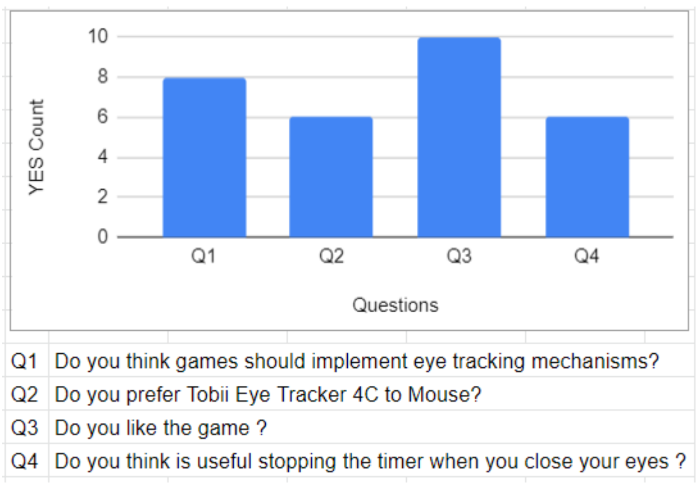


Figure 2.9: *Tobii Eye Tracker 4C* vs Mouse :: UX and Learning Curves

For a better understating a survey to the 10 individuals was performed, including 4 questions (figure 2.10). In general the use of *Tobii Eye Tracker 4C* wins in all questions. However we didn't reach the 10 "YES" in some questions. Maybe the game could be improved or the calibration of the *Eye Tracker* wasn't correctly done.



Q1	Do you think games should implement eye tracking mechanisms?
Q2	Do you prefer Tobii Eye Tracker 4C to Mouse?
Q3	Do you like the game ?
Q4	Do you think is useful stopping the timer when you close your eyes ?

Figure 2.10: Survey

What's so special about *Tobii Eye Tracker*?

This platform improved the traditional *PCCR* (pupil centre corneal reflection), using an infrared light to illuminate the eye, and causing highly visible reflections. Then the cameras of this platform take high-resolution images of the user's eyes to be used in an image processing algorithms and in physiological 3D models so we can estimate the position of the eye in space and the point of gaze with high accuracy.

Chapter 3

Requirements and Functionalities

After we described the more technical concepts about the theme of this project in previous chapters, it's time to propose a solution, and translate functionalities in a set of use cases associated with functional and non-functional requirements.

3.1 Proposed Solution

Performing complex manufacturing processes while interacting with computer systems having hands occupied it's not easy, and often leads to unwanted failures (chapter 1).

The baseline solution should be the combination of movement, fixation and blinking of the user eyes with hand gestures to do simple operations in *Critical Manufacturing* system (*MES*).

Since in the top of our metrics is the involvement of components with a very high trust rating, the solution should promote the integration of a specialized hardware for eye tracking. In this case the chosen hardware was the *Tobii Eye Tracker 4C*, based on the good characteristics described before, and an *HD 1080p Webcam* incorporated with the *MES* user interface that allows to extract the user hands with good quality.

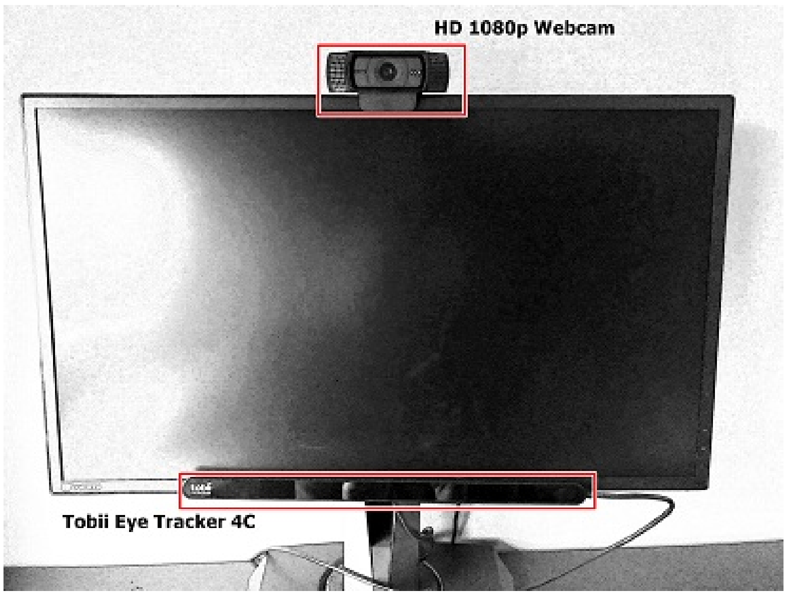


Figure 3.1: Package diagram

Another important point of this project was also the development of machine learning engines that predict eye movements and detect hand gestures, enabling the management and maintenance of multiple dynamic actions inherent to these eye/hand movements.

Also should be possible to record the eye movements associated to some daily operation tasks in order to be able to find possible faults associated with *MES*, as well as check which *Wizards/Execution Views* users spend most time.

3.2 Use Cases

The general requirement of allowing users to interact with the *MES* user forms without using the traditional keyboard and mouse leads to the use of other modalities involving the eyes, hand gestures, and possibly speech.

These three modalities conducted to the grouping of use cases in these three modalities.

The package diagram of figure 3.2 represents the described generic model of use cases for the system developed. Each package, groups one or more parts of the system that are intended to support the organization's processes and/or to gather a set of features. Each package includes the actors and use cases developed for the system.

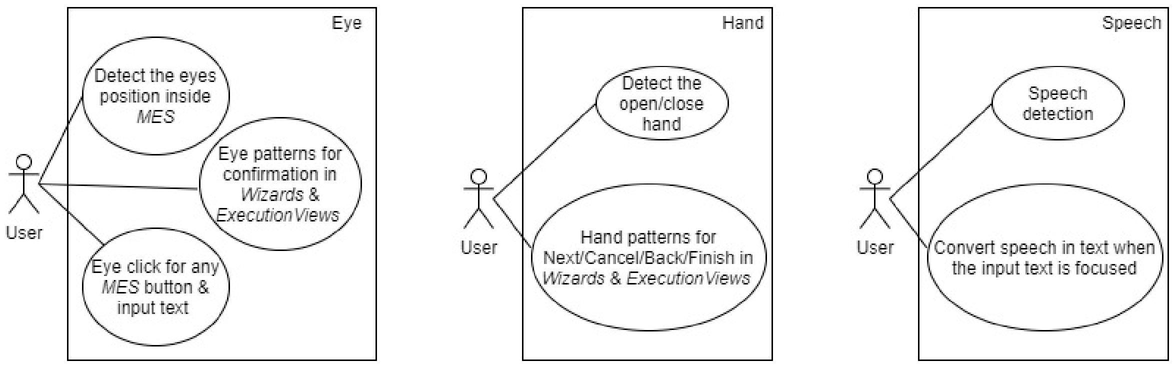

Eye
Detect the eyes position inside MES
Eye patterns for confirmation in Wizards & Execution Views
Eye click for any MES button & input text
User
Hand
Detect the open/close hand
Hand patterns for Next/Cancel/Back/Finish in Wizards & Execution Views
User
Speech
Speech detection
Convert speech in text when the input text is focused
User

Figure 3.2: Package diagram

In table 3.1 the actors of the system are presented, and specifies table 3.2 each of the packages.

Table 3.1: Actors of the system

Actor	Description
User	All individuals that uses the system. Usually, they should be operators that are doing some processes that required the usage of several tasks at the same time, like mounting an hardware component while they are checking the steps presented in *MES*.

Table 3.2: Packages of the system

Package	Description
Eye	Involves all processes associated to the eye detection and interaction.
Hand	Hand detection/interaction and all processes associated.
Speech	Package for speech detection.

The following subsections 3.2.1, 3.2.2 and 3.2.3 describe the use cases for the corresponding package.

The use cases are a list of actions or event steps typically defining the interactions between an actor and a system, to achieve a goal [16]. The actor can be a human or other external system. Each use case is composed with an *ID, Name, Objective, Description, Pre-conditions* and *Normal Flow*.

3.2.1 Package: Eye

Package: Eye includes use cases for detect the eyes position inside *MES*, eye click for any *MES* button and input text, and eye patterns for confirmation in *Wizards* and *Execution Views*.

Tables 3.3, 3.4 and 3.5, represent in detail the use cases associated to *Package: Eye*.

Table 3.3: UC.001 - Detect the eyes position inside *MES*

UC.001	**Detect the eyes position inside *MES***
Objective	Use of an *Eye Tracker* inside *MES*.
Description	When *User* is inside *MES*, he can see to what his eyes are looking at.
Pre-conditions	Every *User* should have an *Eye Tracker* hardware like *Tobii Eye Tracker 4C*.
Normal Flow	1. The *User* accesses to the system. 2. The system starts with a login page. 3. The *User* logs in. 4. The *User* selects the option *Administration* in left panel. 5. The *User* selects the option *Movement Tracking* in central panel. 6. The *User* clicks in button *Movement Tracking Deactivated* in central panel.

Table 3.4: UC.002 - Eye click for any *MES* button and input text

UC.002	**Eye click for any *MES* button and input text**
Objective	Clicks in button/input text, using *Eye Tracker*.
Description	When *User* looks to a button/input text, the inner action, after a little delay, is triggered.
Pre-conditions	Every *User* should have an *Eye Tracker* hardware like *Tobii Eye Tracker 4C*.
Normal Flow	1. Follow the steps of use case *UC.001* (table 3.3). 2. The *User* looks to a button or input text. 3. After a while the border of button or input text changes to yellow then to red. 4. The button or input text is pressed and the inner action is triggered.

Table 3.5: UC.003 :: Eye patterns for confirmation in *Wizards* and *Execution Views*

UC.003	**Eye patterns for confirmation in *Wizards* and *Execution Views***
Objective	Adds an extra validation layer when user finishes a *Wizard/Execution View*.
Description	When *User* finishes a *Wizard/Execution View*, he must do an eye pattern for confirm/cancel his action.
Pre-conditions	Every *User* should have an *Eye Tracker* hardware like *Tobii Eye Tracker 4C*.
Normal Flow	1. Follow the steps of use case *UC.001* (table 3.3). 2. The *User* go to a *Wizard* or *Execution View*. 3. The *User* looks to finish button. 4. After a while, the border of button changes to yellow then to red. 5. An eye confirmation modal should open for user performs an eye pattern. 6. The *User* looks to a central point until is red. 7. The *User* performs an eye pattern and finish with eyes blinking.

3.2.2 Package: Hand

Package: Hand includes use cases for the hand gestures for *Next/Cancel/Back/Finish* in *Wizards/Execution Views*, and the detection of open/close hand.

We can see in Tables 3.6 and 3.7, the use cases details associated to *Package: Hand*.

Table 3.6: UC.004 :: Hand gestures for *Next/Cancel/Back/Finish* in Wizards and Execution Views

UC.004	**Hand gestures for *Next/Cancel/Back/Finish* in *Wizards/Execution Views***
Objective	Performing the *Next/Cancel/Back/Finish* easily, with a single hand gesture.
Description	The *User* shows his hand to the webcam, performing a specific hand gesture that represents an operation in *Wizard/Execution View*.
Pre-conditions	Every *User* should have a webcam correctly connected.
Normal Flow	1. Follow the steps of use case *UC.001* (table 3.3). 2. The *User* go to a *Wizard/Execution View*. 3. The *User* blinks his eyes for starting hands gesture detection. 4. The *User* should close his hand to tell the system that an hand gesture is starting. 5. The *User* opens his hand and make an hand gesture. 6. The *User* finishes the hand gesture when he closes his hand. 7. The system will triggered an event of *Next/Cancel/Back/Finish* depending on the hand gesture performed by the *User*.

Table 3.7: UC.005 :: Detect the open/close hand

UC.005	**Detect the open/close hand**
Objective	*User* hands detection can be used to do operations in *MES*.
Description	Detects the *User* hand, when shown on the webcam.
Pre-conditions	Every *User* should have a webcam correctly connected.
Normal Flow	1. The *User* should run the application for hand detection. 2. The *User* sees that his hand is detected when he shows it to the webcam.

3.2.3 Package: Speech

Package: Speech includes use cases for speech detection.

Tables 3.8 and 3.9, describe the use cases details associated to *Package: Speech*.

Table 3.8: UC.006 :: Speech detection

UC.006	Speech detection
Objective	Detects the *User* speech.
Description	Speech detection is very important when we want to set text in the input components without typing.
Pre-conditions	The *User* should have a microphone.
Normal Flow	1. The *User* accesses to the system. 2. The system starts with a login page. 3. The *User* logs in. 4. The *User* selects the option *Administration* in left panel. 5. The *User* selects the option *Movement Tracking* in central panel. 6. An browser pop-up should automatically open, requesting the *User* to add permission to his microphone.

Table 3.9: UC.007 :: Convert speech in to text when the input text is focused

UC.007	**Convert speech in to text when the input text is focused**
Objective	Writes input text of the speech that *User* is speaking
Description	Converts speech into text and add value to input text.
Pre-conditions	The *User* should have a microphone.
Normal Flow	1. Follow the steps of use case *UC.006* (table 3.8). 2. The *User* must be focused in an input text. 3. The *User* should speak to microphone. 4. The speaking is converted to text and added into the current focused input text.

3.3 Functional Requirements

The functional requirements define the basic system behaviour, they contain what the system does or must not do. Each functional requirements is composed with an *ID, Name, Priority, Description* and *Motivation*.

For this project, the functional requirements specify the use of *MES* with eyes, patterns with eyes, eye patterns customization, perform operations using hand gestures, and hand gestures customization.

Tables 3.10 to 3.14 describe in more detail each of these requirements.

Table 3.10: REQ.001 :: Using *MES* with eyes

REQ.001	**Using *MES* with eyes**
Priority	Essential.
Description	Possibility to control *MES* using an *Eye Tracking* solution.
Motivation	It's common to see operators wearing gloves, maneuvering heavy or sensitive material, performing complex assembly processes while interacting with computer systems, having their hands occupied. Using an *Eye Tracking* solution will improve this interaction with *MES*

Table 3.11: REQ.002 :: Patterns with eyes

REQ.002	**Patterns with eyes**
Priority	Essential.
Description	Using extra pattern validation to confirm operations.
Motivation	Important finishing operations in *Wizards* and *Execution Views*, like track-in/track-out a material, must require an extra validation, using an eye pattern to confirm/cancel.

Table 3.12: REQ.003 :: Eye patterns customization

REQ.003	**Eye patterns customization**
Priority	Essential.
Description	The solution must have the possibility to create/update eye patterns.
Motivation	Eye patterns, by default should have a specific pattern, that eventually can be customized by *CMF* clients.

Table 3.13: REQ.004 :: Perform operations using hand gestures

REQ.004	**Perform operations using hand gestures**
Priority	Essential.
Description	Possibility to use hand gestures in *Wizards* and *Execution Views*.
Motivation	This functionally can be described as a set of shortcuts using hand gestures, that corresponds to inner operations in *Wizards* and *Execution Views*, like *Next/Back/Finish/Cancel*.

Table 3.14: REQ.005 :: Eye patterns customization

REQ.005	**Hand gestures customization**
Priority	Essential.
Description	The solution must have the possibility to create/update hand gestures.
Motivation	Hand gestures, by default should have a specific pattern, that eventually can be customized by *CMF* clients.

3.4 Non Functional Requirements

The non functional requirements specifies criteria that can be used to judge the operation of a system, rather than specific behaviours, they are constraints on development that limit some degree of design freedom for those building the system. Each non functional requirements is also composed with an *ID, Name, Priority, Description* and *Motivation*.

In our case the solution should contemplate *GUI* according to *MES*, fast performance and optimization, system error log, avoiding additional dependencies, using *CMF Message Bus* for multicast communication and code clean and maintainable.

Tables 3.15 to 3.20 show all the details contemplated on these kind of requirements.

Table 3.15: REQ.006 :: *GUI* according to *MES*

REQ.006	***GUI* according to *MES***
Priority	Essential.
Description	*CMF* has his own image, is essential that guidelines are followed.
Motivation	*CMF* have his specific *GUI* components, it should be avoided the creation of visual components that uses different style.

Table 3.16: REQ.007 :: Fast performance and optimization

REQ.007	**Fast performance and optimization**
Priority	Essential.
Description	Build modules with fast performance and optimization.
Motivation	Since *CMF* operates in critical systems, it should be paid attention to detail, taking care about the performance using guidelines to get the best functionality.

Table 3.17: REQ.008 :: System error log

REQ.008	**System error log**
Priority	Essential.
Description	When something goes wrong, the system should save a log file with errors.
Motivation	For a good traceability, when system has some error, the client can send the log file to *CMF* support team to care of the problems.

Table 3.18: REQ.009 :: Avoiding additional dependencies

REQ.009	**Avoiding additional dependencies**
Priority	Essential.
Description	Third-party dependencies should be avoided.
Motivation	*CMF* as a set of solutions for the majority of the problems, an additional dependency should be avoided and if it's really needed we should think in these questions: How popular is the library ? How reliable is the author ? How well-written is the library ? Does the library meet your specific requirements ? Does the library have the correct license ? Is the library open source ? Is the library recommended ? Is this library going to be used for core features ?

Table 3.19: REQ.010 :: Using *CMF Message Bus* for multicast communication

REQ.010	**Using *CMF Message Bus* for multicast communication**
Priority	Essential.
Description	The communication between *Eye Tracker*, webcam and *MES* should use *CMF Message Bus*.
Motivation	*CMF* has is own *Message Bus* to communicate multicast messages between systems, this project should use the same guidelines.

Table 3.20: REQ.011 :: Code clean and maintainable

REQ.011	**Code clean and maintainable**
Priority	Essential.
Description	The development must be commented and perfectly understandable for a better maintenance.
Motivation	This project should be extensible for another requirements or enhancement, it's very important that code is commented and perfectly understandable .

Chapter 4

Architecture

The requirements listed in the previous chapter are mostly related to the user interface as presented by the *MES* software.

For the correct adding and implementation of those requirements it is necessary not only to study the architecture of the *MES product* from *Critical Manufacturing*, but also understanding some important business key factors.

The *CMF Product* is designed for scalability and high-availability and is able to run in a single computer or in a very distributed server farm as illustrated in figure 4.1.

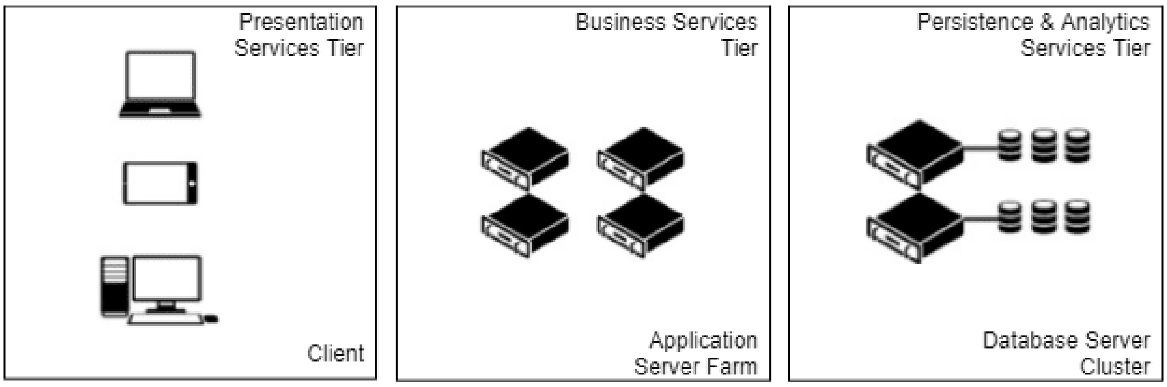


Figure 4.1: Product Architecture

In general, the three most important layers are:

- **Presentation Services**
  The *GUI* runs in the client, although the *HTML* must be served by a Web Server such as *IIS*. Regarding high availability, a failure in a single client does not compromise the availability of other clients or of any other component of the system.
- **Business Services**
  Using the *CMF Load Balancing* module, application server nodes can be added (scale

out) and removed dynamically without system downtime. It also provides high availability. Adding more powerful hardware resources (scale up), such as adding more memory is also possible and beneficial for the application performance.

- **Persistence and Analytics Services**
  SQL Server Always On with Read-Only Replica – allows read-only statements to hit the secondary database server. *CMF* Database Distribution uses three databases : *Online*, *Operational Data Store (ODS)* and a *Data Warehouse (DWH)*. The databases can be placed in three different *SQL Server* instances running in different servers.

One of the biggest concerns of *Critical Manufacturing* is the need of fast solutions, avoiding the usage of additional dependencies, like new external *npm*[1] modules or another stack, different from the *MES product* stack. Considering these prerequisites, it was decided to use the *Message Bus* already developed by *CMF*, instead of a queue solution, such as *rabbitMQ*, for communication between modules. The programming languages used for the project architecture are *Typescript* (using *Angular* as front-end framework), *LESS*, *HTML* for front-end tier and *C#* using *.NET Framework* and *.NET Core 2.0* for back-end tier.

This architecture has 3 important modules shown on figure 4.2. The first module, represents all *GUI* tier (*MES*). The second module handles the problems of *Eye Tracking* itself, analyse the eye coordinates and predict actions with help of a machine learning engine (*EyeTracking Middleware*). The third module handles hand gestures and also has a machine learning engine to predict the *User* actions (*HandTracking Middleware*). The *EyeTracking Middleware* and *HandTracking Middleware* modules can run independently, we can use both or one, depending on customer needs.

The *EyeTracking Middleware* and *HandTracking Middleware* are directly related to the use of specific hardware. In case of *EyeTracking Middleware* we only can use *Tobii Eye Tracker 4C* device, to collect all the user's eyes data, while in the case of *HandTracking Middleware* we can use any kind of webcam.

[1] `https://www.npmjs.com/`

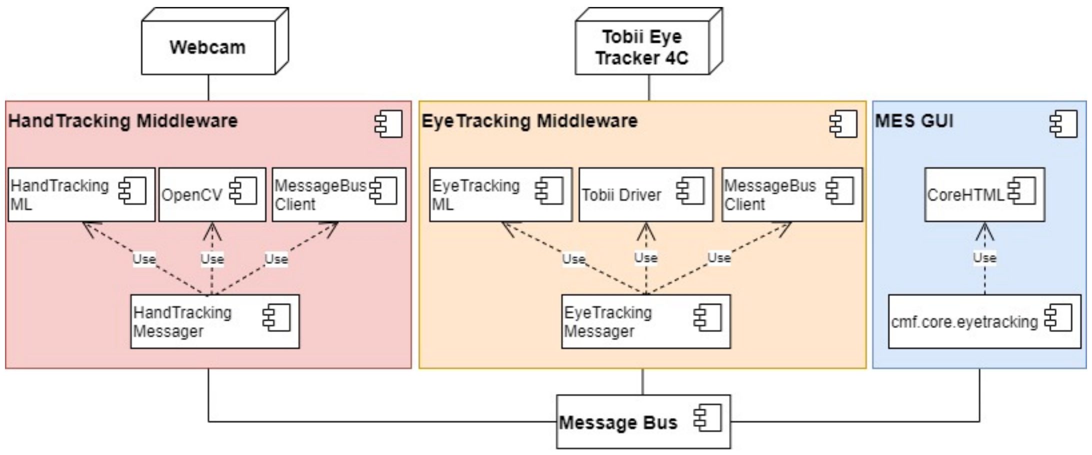


Figure 4.2: Detailed Eye Tracking Components Diagram

4.1 MES GUI

This module represents the front-end tier, that evolves the *MES* and *CORE* components already developed and in operation for *Critical Manufacturing* with a new customization sub-module named *cmf.core.eyetracking*, that corresponds to the work of this project, *Eye/Hand Tracking* and *Speech Recognition*.

The *cmf.core.eyetracking* sub-module, was developed based on the *Angular Architecture Patterns and Best Practices* [31]. As we can see in figure 4.3 we have the *Presentation Layer* when all *Angular Components* lives, that represents the *UI* and delegates user's actions to the *Core Layer*, through the *Abstraction Layer*. The *Abstraction Layer* decouples the *Presentation Layer* from the *Core Layer*. In the *Core Layer* we have all data manipulation and outside world communication.

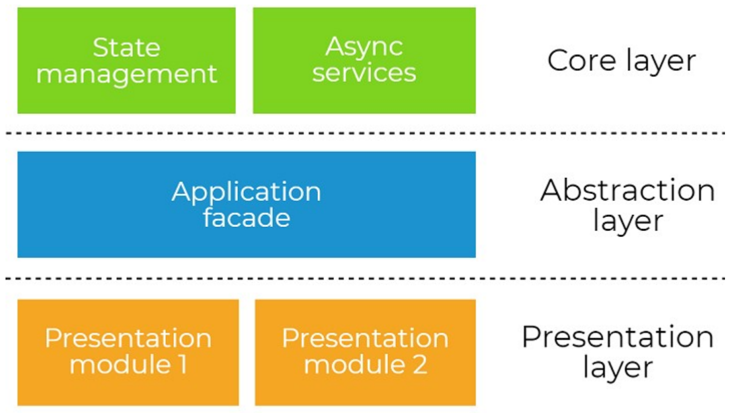


Figure 4.3: High-level abstraction layers

In object-oriented development it's also important to have strategies for flexibility and maintainability of the solution, so it was decided to apply the *SOLID* software designs pattern, announced by *Robert C. Martin* [24] that having 5 principles:

- **SRP - Single Responsibility Principle**
  An active corollary of *Conway's* law, that says the best structure for a software must be highly influenced by the social structure of the organization that uses it, so that each software module has one, and only one, reason to change [9].
- **OCP - Open-Closed Principle**
  Software systems to be easy to change, must be designed to allow their behaviour to change by adding new code, instead of changing existing code. *Bertrand Meyer* popularized this principle in the 80s [25].
- **LSP - Liskov Substitution Principle**
  The concept of this principle was introduced by *Barbara Liskov* in a 1987 saying that child classes should never break the parent class type definitions [22].
- **ISP - Interface Segregation Principle**
  This principle is very related to each interface and mandates that each one should provide a single behaviour. The result should be smaller and more specific interfaces.
- **DIP - Dependency Inversion Principle**
  The code that implements an high-level policy shouldn't depend on the code that implements lower-level details, but instead depend on an abstraction.

Following the previous principles we have added for the *Presentation Layer* the *Angular Components pageMovementTracking*, *movementSettings* and *eyeAction* that represents the *UI* associated to *Eye/Hand Tracking* and *Speech Recognition*. Then for *Abstraction Layer*, it was used *Dependency Inversion* to separate the *Angular* services (*EyeTrackerEventsService, HandTrackerEventsService, SpeechRecognitionEventsService*) by abstraction and *Dependency Injection* to eliminate the manual instantiation as *Inversion of Control (IoC)* abstract programming principles says. At last, the *Core Layer* contains the observers and async events related to *Message Bus* activities and *DOM* events.

4.2 Eye Tracking Middleware

Eye Tracking Middleware is the core module for the eyes detection, which has two solutions. One directly related to the connection to the *Eye Tracker* hardware, and another that predicts the *Eye Actions* with the help of a *Machine Learning* engine.

As we can see in figure 4.4, each client has one *Tobii Eye Tracker 4C* that communicates directly to the *Eye Tracking Middleware*. In this last module, the gaze points that came from the tracker hardware are processed, analyzed and sent by the *Message Bus*, together with the *Eye Actions* predicted by the *ML Eyes Engine*.

Fortunately, the *Tobii Eye Tracker 4C* hardware has an *API* named *Tobii.Interaction*, that treats the integration of the hardware with our middleware, and has some events associated to the eyes detection and gaze points collection.

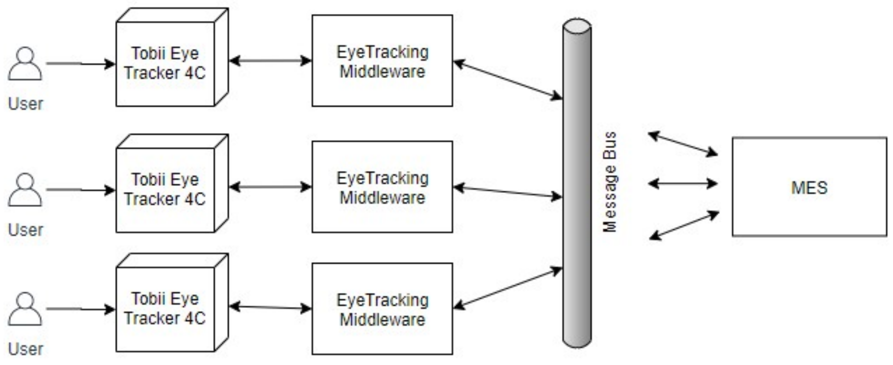


Figure 4.4: Eye Tracking Components Diagram

Tobii's eye detection follows a set of steps, but unfortunately the actual technique explored by the device is not by the manufacturer. However it was assumed that the image processing required for gaze data calculations is performed by the *Tobii EyeChip*, located on the *Eye Tracker*, that contains all *Tobii* algorithms necessary for *Eye Tracking* computation. The *Tobii EyeChip* reduces power consumption, CPU load, and data transfer between the *Eye Tracker* and its host computer.

The *Tobii Eye Tracker 4C* is connected via a *USB 2.0 port*, working at a distance of 0.5 m to 0.95 m, and has an incredible data rate of 90 Hz, which means the *Tobii Eye Tracker 4C* tracks where we are looking 90 times per second. The maximum recommended screen size is 0.69 m with 16:9 aspect ratio or 0.76 m with 21:9 aspect ratio.

Considering an user positioned at the far limit of the operating distance (800 mm), the working range of the device in degrees of visual angle is [- 18º, 18º] on the x-axis, and [- 10.5º, 10.5º] on the y-axis. For instance, at a distance of 700 mm, the users may move their head 240 mm leftwards or rightwards and 195 mm upwards or downwards [2].

4.3 Hand Tracking Middleware

Similar to the *Eye Tracking Middleware*, this module is composed by two solutions. One that is responsible for getting, processing and analysing the images of the hands that comes from the webcam, and another that predicts the hand actions with the help of a machine learning engine.

In figure 4.5, as we can see, each client has one webcam that communicates directly to the *Hand Tracking Middleware*. There the *Cascade Classifiers* with *Multi Scale Detection* are enabled to know the central point of the hand and sends that coordinates to *Message Bus*. Also, the *Hand Actions* predicted by the *ML Hands Engine* are sent to the *Message Bus*.

The *ML Hands Engine* is responsible to detect gestures that corresponds to the *Next / Cancel / Back / Finish* buttons in *Wizards* and *Execution Views*, which will be described in the next chapter.

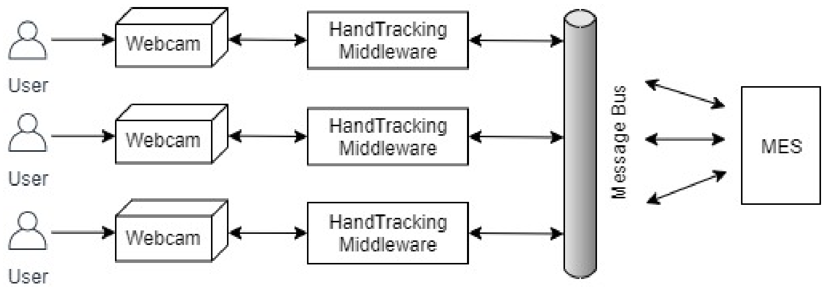


Figure 4.5: Hand Tracking Components Diagram

4.4 Message Bus

For sending broadcast messages the *MES* has a specific component with a high-performance publish/subscribe message bus that implements a subject-based address system.

The *Message Bus* is a combination of a common data model, a common command set, and a messaging infrastructure that allows different systems to communicate through a shared set of interfaces. There may be no guarantee of first-in-first-out ordering, and the *Message Bus* subscribers can come and go without the knowledge of message senders. Unlike queues, where the sending application explicitly adds messages to every queue, *Message Bus* uses a publish/subscribe model. Messages are published to the *Message Bus*, and any application that has subscribed to that kind of message will receive it. This approach allows applications to follow the open/closed principle [24], since they become open to future changes while remaining closed to additional modification.

In order to use *Message Bus* features, first we must set the transport configuration, like ports and addresses of *Gateway* and *Load Balancer* components, then a channel for our specific context. The *Connected*, *Disconnected*, *Exception*, *InformationMessage* events must be implemented, since they give us important information regarding the state of the *Message Bus*.

After starting the *Message Bus* client, we have the option to publish or subscribe new messages, depending the operation that we want to do. In the tables 4.1, 4.2, 4.3, 4.4 and 4.5 the defined and implemented types of messages and it usage are shown.

Table 4.1: *Hand Tracking Middleware* - Messages Send/Receive

Subject	**Message**	**Type**	**Description**
HAND_ACTION	*"Action: {0}"*	Send	Hand action.
EYES_BLINKING	*"Blinking: true"*	Receive	When user blink his eyes.

Table 4.2: *Eye Tracking Middleware* - Messages Sent

Subject	**Message**	**Description**
EYE_MESSAGE	*"X: {0}, Y: {1}"*	Sends coordinates *[X,Y]* of the eyes in screen.
EYE_TRACKER_CONF	*"TrackerConfig: {0}"*	Sends *Eye Tracker* configuration.
EYE_ACTION_RESPONSE	*"Action: {0}"*	Sends *Eye Action*.
EYES_BLINKING	*"Blinking: true"*	Sends user blinking.

Table 4.3: *Eye Tracking Middleware* - Messages Received

Subject	**Message**	**Description**
EYE_REQUEST_TRACKER_CONF	*"TrackerId: {0}"*	Event receive for *Eye Tracking* configuration.
EYE_UPDATE_TRACKER_CONF	*"TrackerConfig: {0}"*	Event receive for updating *Eye Tracking* configuration.
EYE_TRACKING_START	*"TrackerId: {0}"*	Event receive for starting the *Eye Tracker*.
EYE_TRACKING_STOP	*"TrackerId: {0}"*	Event receive for stopping the *Eye Tracker*.
EYE_ACTION_REQUEST	*"Xf_Xi: {0}, Yf_Yi: {1}"*	Event receive for predict the *Eye Action*.

Table 4.4: *MES* - Messages Sent

Subject	**Message**	**Description**
EYE_REQUEST_TRACKER_CONF	*"TrackerId: {0}"*	Request *Eye Tracking* configuration.
EYE_UPDATE_TRACKER_CONF	*"TrackerConfig: {0}"*	Sends *Eye Tracking* configuration for updated.
EYE_TRACKING_START	*"TrackerId: {0}"*	Sends message for starting the *Eye Tracker*.
EYE_TRACKING_STOP	*"TrackerId: {0}"*	Sends message for stopping the *Eye Tracker*.
EYE_ACTION_REQUEST	*"Xf_Xi: {0}, Yf_Yi: {1}"*	Sends message for predict the *Eye Action*.

Table 4.5: *MES* - Messages Received

Subject	**Message**	**Description**
EYE_TRACKER_CONF	*"TrackerConfig: {0}"*	Receives *Eye Tracker* configuration.
EYE_ACTION_RESPONSE	*"Action: {0}"*	Receives *Eye Action*.
EYES_BLINKING	*"Blinking: true"*	Triggered when user blinking his eyes.

Chapter 5

Implementation

After describing the main features of the software architecture for incorporating the new functionality to be available in the *MES* application, we now focus in some important details of their implementation.

5.1 Overlap Detection

When the eye point coordinates *[X,Y]* is transmitted to the *MES GUI*, through *Message Bus*, we must know what object we are looking at on the screen. We consider that a user is looking not for a specific point, but for a set of points centered in those coordinates and with a threshold radius. By default that radius has a customizable value of 30 pixels, so we can improve the accuracy and reduce the *Eye Tracker* error.

In *MES GUI*, for reducing the DOM search, we consider only the components that make sense to be tracked, which are: *BUTTON, INPUT, CMF-CORE-CONTROLS-ACTIONBUTTON, CMF-CORE-CONTROLS-CONTEXT-MENU, CMF-CORE-CONTROLS-PANELBAR, and menu-toggle*.

Lets imagine that object A is a trackable DOM element, and object B is the point (with a tolerance radius and a circumscribed square) that we are looking at the moment (see figure 5.1).

How can we know that these objects are overlapping ? We know by Morgan's law that:

> The negation of a disjunction is the conjunction of the negations;
>
> The negation of a conjunction is the disjunction of the negations;

Or

> The complement of the union of two sets is the same as the intersection of their complements;
>
> The complement of the intersection of two sets is the same as the union of their complements;

Or

$\overline{A \cup B} = \overline{A} \cap \overline{B}$;

$\overline{A \cap B} = \overline{A} \cup \overline{B}$;

Where A and B are sets:

A is the complement of A;

$\cap$ is the intersection;

$\cup$ is the union.

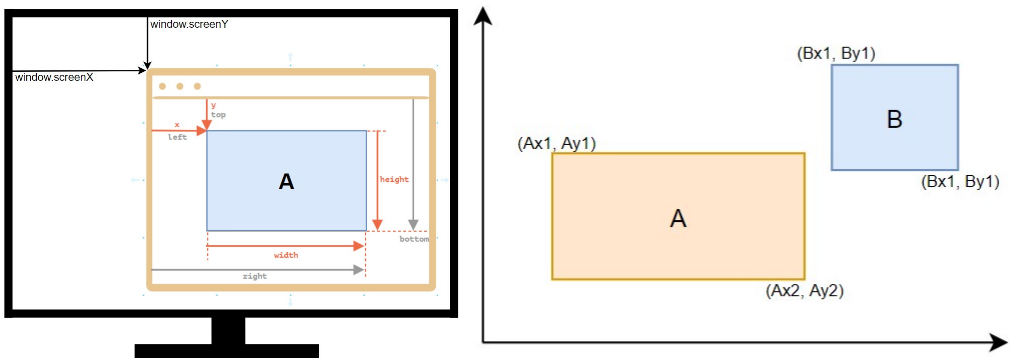


Figure 5.1: Coordinates Mapping and Collision between two objects

Cond.1: If A's left edge is to the right of the B's right edge - Then A is Totally to right Of B;

Cond.2: If A's right edge is to the left of the B's left edge, - Then A is Totally to left Of B;

Cond.3: If A's top edge is below B's bottom edge - Then A is Totally below B;

Cond.4: If A's bottom edge is above B's top edge - Then A is Totally above B.

So condition for Non-Overlap is *Cond.1* Or *Cond.2* Or *Cond.3* Or *Cond.4*

Therefore, a sufficient condition for Overlap is the opposite (De Morgan).

We prove by contradiction of Morgan's law that overlap can exist if the opposite of all these conditions are satisfied.

5.2 Eye Coordinates

As shown in the figure 5.2, the process of getting the coordinates that correspond to where the user is looking, is associated with a set of events.

We start to select, in the front-end tier, what *Eye Tracker Hardware* we want to use (that event is associated to a *Message Bus* message named *EYE_TRACKING_START*.

When *EyeTracking Middleware* is on the start state, it sends a *Message Bus* message named *EYE_TRACKER_CONF*, with the *Eye Tracker Configuration*. Then, 90 times per second, the *EyeTracking Middleware* sends the *[X,Y]* coordinates, previously filtered, that corresponds to where the user is looking at.

Now that we have the coordinates relative to the screen, we must translate that coordinates relatively to the browser window, and for that we must know the *window.ScreenX* and the *window.ScreenY*. After, a customizable radius around the point of should be set, instead of *[X,Y]* coordinates.

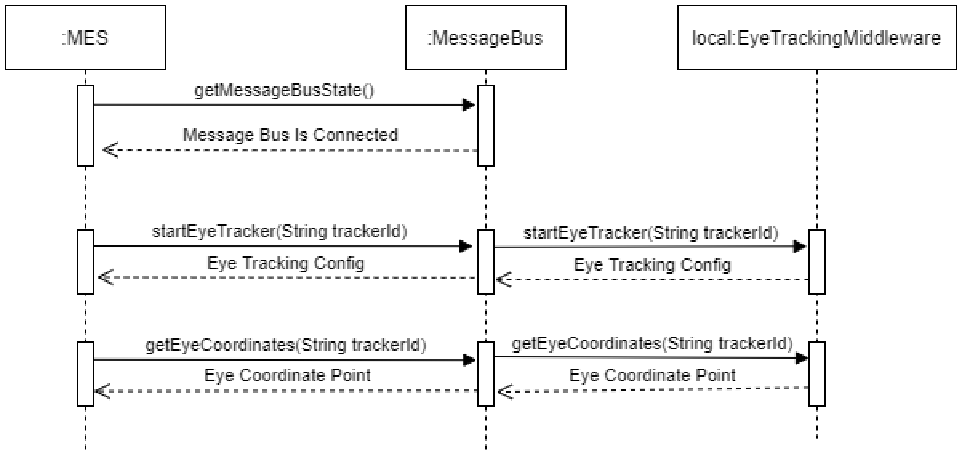

:MES
:MessageBus
local:EyeTrackingMiddleware
getMessageBusState()
Message Bus Is Connected
startEyeTracker(String trackerId)
Eye Tracking Config
startEyeTracker(String trackerId)
Eye Tracking Config
getEyeCoordinates(String trackerId)
Eye Coordinate Point
getEyeCoordinates(String trackerId)
Eye Coordinate Point

Figure 5.2: Get *Eye Tracking* Coordinate

5.3 Machine Learning Pipelines

Three *ML* different pipelines, but in general the steps are similar. We have one *ML* pipeline to detect the human hand, other to detect the eyes movement that will represent *Eye Actions*, and a third for hand gestures that will represent hand actions.

The *ML* pipelines follow the steps: *Build Model, Train Model, Evaluate Model, Consume Model* as shown on figure 5.3 [27].

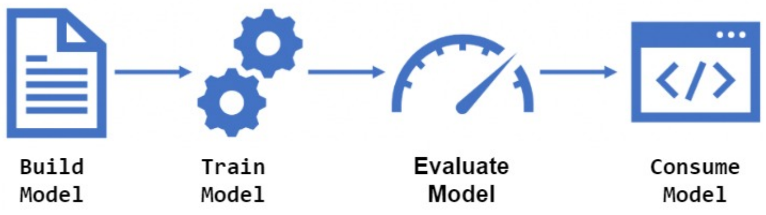


Figure 5.3: Machine Learning Pipeline

In the *Build Model* step, we begin to setup common data loading configuration and process configuration with pipeline data transformations. In the case of eyes movement and hand gestures, the data source is a text file with multiple columns, that have *[X,Y]* coordinate points and one column for the associated action. For hand detection, the data source is a set of negative and positive hand images.

The first step finishes, by defining a data training algorithm, that in case of the eyes movement is *Linear SVM*, of hand gestures is *Average Perceptron* and hand for detection is *Haar Cascade*.

In *Machine Learning*, training data is the key factor to make the machines recognize the objects or certain patterns and make the right prediction when used in real-life.

The *Train model* step is a process of running the chosen algorithm on a training data to tune the parameters of the model. This step can take a while depending of the data complexity and the number of iterations.

After the model is trained, we need to conclude how accurate our model operates on new data (*Evaluate Model* step). For doing this, the model from the previous step runs against another dataset that was not used in training. In case of eyes movement and hand gestures the dataset for evaluation is a set of lines with multiple columns, that have *[X,Y]* coordinate points. In case of hand detection, an hand image that was not used for training was used. *Evaluate Model* calculates the difference between known types and values predicted by the model in various metrics.

We end the *ML Pipeline* generating an *ML* model that can be used for this implementation (the *Consume Model*).

5.4 Binary Classification With Linear SVM for Eye Actions

Since we are clicking in the *CMF MES DOM* components with our eyes, a new requirement has emerged, which is the fact that we must have an extra validation to all finishing operations in *Wizards* and *Execution Views*.

The user must be aware that he wants to do the operation because we don't want a material to be finished by mistake or a resource to be placed in a state that is not supposed to.

One of the possible solutions for this extra validation could be putting the user to make a pattern with his eyes in case he wants to cancel, or continue, the action (*Eye Action*). First, the user looks during *1s* to a central point (telling the system that he's starting the pattern), and when he finishes the action, he must blink his eyes twice (telling the system that no more points should be analyzed and the pattern was finished).

So, it was decided to develop a generic behavior to all finishing operations in *Wizards* and *Execution Views*, asking the user to make a pattern with his eyes, in case he wants to cancel, or continue, the action (*Eye Action*).

To perform this action, a *ML* engine was developed, detecting a pattern to decide what action the user wants to do. For acquiring the needed data a sequence of steps should be performed, as we can see in figure 5.4.

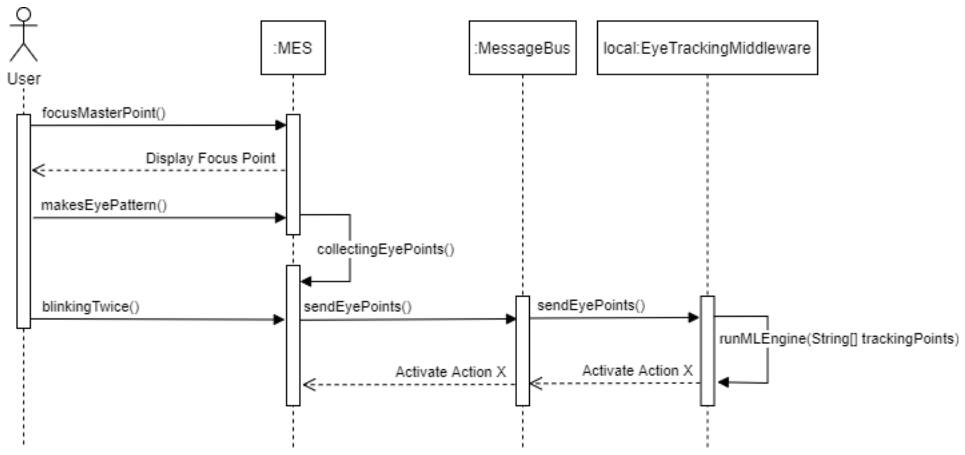


Figure 5.4: *Eye Action* Prediction

The *Machine Learning* engine for *Eye Actions* is based on *Linear SVM* (this decision will be discussed in the chapter 6, subsection 6.2.2).

The algorithm finds a hyperplane in the feature space for binary classification, by solving a *SVM* problem. For instance, with feature values $f_0, f_1, ..., f_{D-1}$, the prediction is analysing what side of the hyperplane the point falls into [26].

The *ML Pipeline* for training and predicting *Eye Actions* (section 5.3) has the input formula :

$$\begin{aligned}
&\vec{v} = [a, b, action] \\
&a = Y_{final} - Y_{initial} \wedge b = X_{final} - X_{initial} \wedge action = 0 \vee 1 \qquad (5.1) \\
&Y_{final}, Y_{initial}, X_{final}, X_{initial} \in R
\end{aligned}$$

We start with a pre-processing of an array with multiple points of *[X,Y]* coordinates, that represents the movement that user made with his eyes. Then the values *a*, *b* and *action* can be calculated, and added at $\vec{v}$.

The output data has two properties, the action (boolean) and the score (float). If the action is false, we want to proceed the operation, if is true we want to cancel (in the figure 5.5 we can see the configurable *Eye Actions* by default). The score property shows us the certainty value with which the prediction was made.

Eye Actions are totally full customizable by the user, as long as follow a set of *ML* rules.

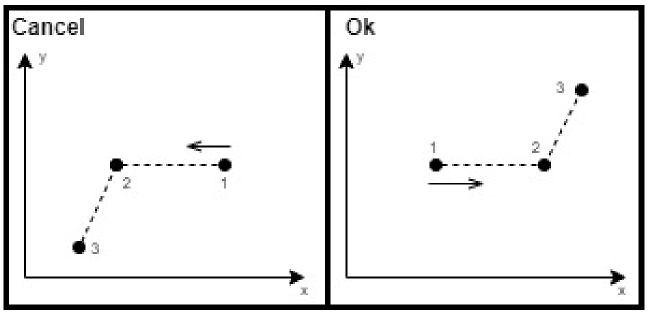


Figure 5.5: *Eye Actions*

5.5 Multi Classification With Average Perceptron for Hand Actions

Sometimes, in the daily operations it's helpful to have the possibility to define shortcuts in actions, using hand gestures and avoiding the usage of mouse or keyboard. The idea was allowing a simple configurable hand gesture to trigger the button *Back or Cancel or Next or Finish* (hand action) that are present in *Wizards* and *Execution Views*, since they represent the most part of the operators day.

Translating this problem into a solution will requires a *ML engine* specialized in multi classification problems, predicting actions for the user hand gestures.

The algorithm that most closely matches our structure and had the best results is the *Average Perceptron* (the results and the algorithm decision will be discussed in chapter 6, subsection 6.2.1).

The *Average Perceptron* is an extension of basic *Perceptron*, introduced by *F. Rosenblatt* [34] in 1958, when the formula can be given by [5]:

$$f(x,\theta,\theta_0) = sign(\theta * x + \theta_0) \tag{5.2}$$

In our case the features are 3 *[X,Y]* coordinates, that represents the starting, middle and finishing points $(x_1, y_1, x_{mid}, y_{mid}, x_2, y_2)$, where θ is the weight vector, θ_0 is the bias and x is the vector of features.

The *sign* function is used to distinguish x as either a positive or a negative label, where the decision boundary to separate the data with different labels, occurs at:

$$\theta * x + \theta_0 = 0 \tag{5.3}$$

After decision boundary, the hyperplane separates the space into two regions:

$$\begin{aligned} Positive \quad if \quad \theta * x + \theta_0 > 0 \\ Negative \quad if \quad \theta * x + \theta_0 < 0 \end{aligned} \tag{5.4}$$

The data are linearly separable if exists a θ and a θ_0 that $y_i(\theta * x_i + \theta_0) > 0$ for all i points, where y_i is labelled.

The figure 5.6 illustrates an example of the mentioned concepts, where the $x = [x_1, x_2]^T$, $\theta = [\theta_1, \theta_2]$ and θ_0 is a offset scalar.

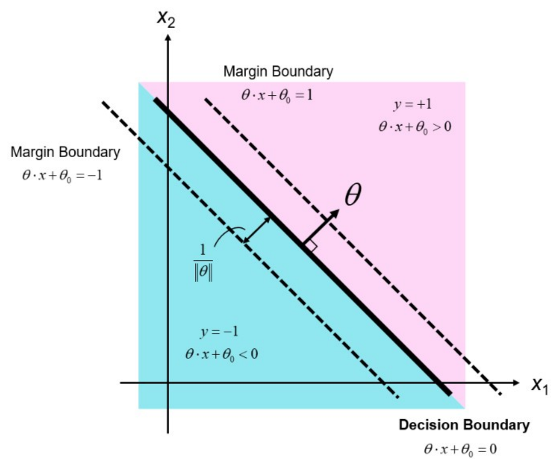

x_2
Margin Boundary
$\theta \cdot x + \theta_0 = 1$
$y = +1$
$\theta \cdot x + \theta_0 > 0$
Margin Boundary
$\theta \cdot x + \theta_0 = -1$
θ
$\frac{1}{\|\theta\|}$
$y = -1$
$\theta \cdot x + \theta_0 < 0$
x_1
Decision Boundary
$\theta \cdot x + \theta_0 = 0$

Figure 5.6: Binary linear classifier example

Hand actions are totally full customizable per user, as long as they follow a set of *ML* rules. As we can see in image 5.7, the actions *Back, Finish, Next, Cancel*, have been defined as the following patterns:

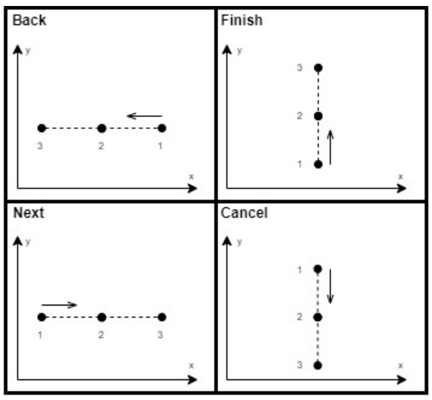


Figure 5.7: Hand Actions

5.6 Haar Cascade for Hand Detection

Hand detection using a webcam is a process that can be tricky and takes it's time, but is crucial for hand actions mentioned in section 5.5.

The chosen approach was the usage of *Haar Cascade* algorithm, that uses machine learning to identify objects in image/video, proposed by *Paul Viola* and *Michael Jones* in paper *Rapid Object Detection using a Boosted Cascade of Simple Features* in 2001 [30].

This algorithm is based on the *Haar Wavelet* technique to analyse pixels in the image into squares by function, and also uses *Integral Images* concepts to compute the objects detected. However, to have more efficient results, we must use *Ada-boost* learning algorithm, which selects a small number of important features from a large set, removing redundant features.

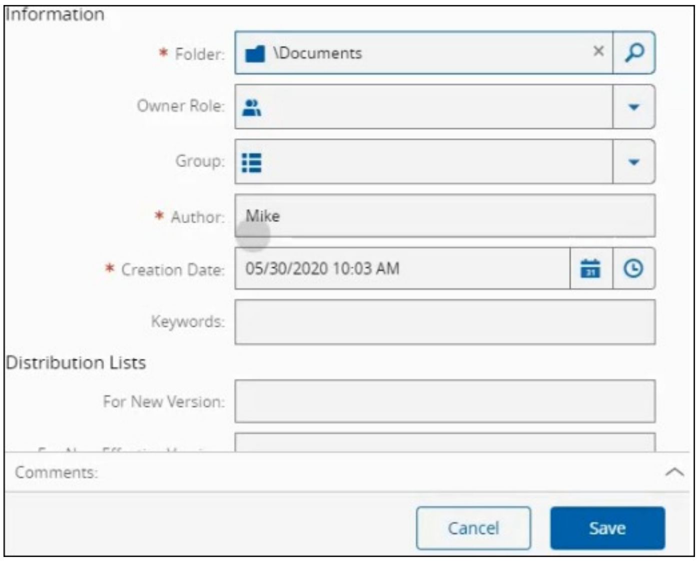
Given an input hand image and convolution kernel, we place the kernel to a corner and do convolution multiplication shifting the kernels. This is like *Convolutional Neural Networks (CNN)*, except that in a *CNN*, the values of the kernel are determined by training, while a *Haar* is manually selected.

When *Haar* features are applied to an hand image, each feature results in a single value which is calculated by subtracting the sum of pixels under a white rectangle from the sum of pixels under a black rectangle. In this case, the white rectangles (relevant areas) could represent the fingers of an hand, and the black rectangles could represent the neighbouring of the fingers.

5.7 Speech Recognition

Speech recognition involves receiving speech through a microphone, which is then checked by a speech recognition service against a list of grammar. When a word or phrase is successfully recognised, it is returned as text string.

Although the speech recognition is not the core of this project, is useful that we have the possibility to set text in the input components without typing, like in the images 5.8 and 5.9.


Information

* Folder:	\Documents
Owner Role:	
Group:	
* Author:	Mike
* Creation Date:	05/30/2020 10:03 AM
Keywords:	

Distribution Lists

For New Version:	

Figure 5.8: Wizard fields

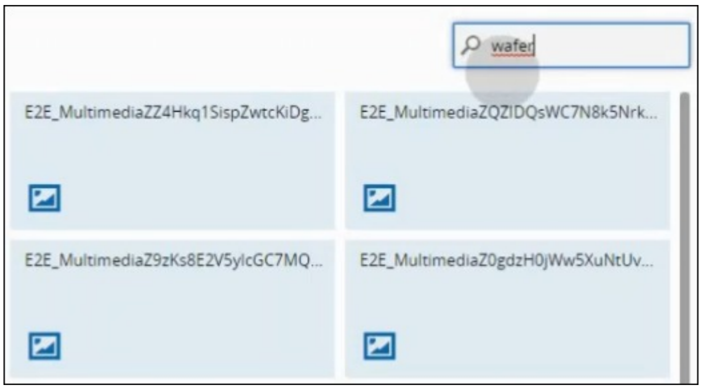

wafer
E2E_MultimediaZZ4Hkq1SispZwtcKiDg...
E2E_MultimediaZQZIDQsWC7N8k5Nrk...
E2E_MultimediaZ9zKs8E2V5ylcGC7MQ...
E2E_MultimediaZ0gdzH0jWw5XuNtUv...

Figure 5.9: Searching in *Tiles Grid*

In this project was used the native *Web Speech API* developed by *Mozilla*.

In figure 5.10, we see the *Mozilla Speech-To-Text Cloud Architecture*, that starts with the sending of data to *Speech-Proxy* to strip the information of the user. Then a request is made to the *STT* provider set in the proxy configuration file, containing only the audio file.

At last, the *STT* provider returns the request containing a transcription to the client.

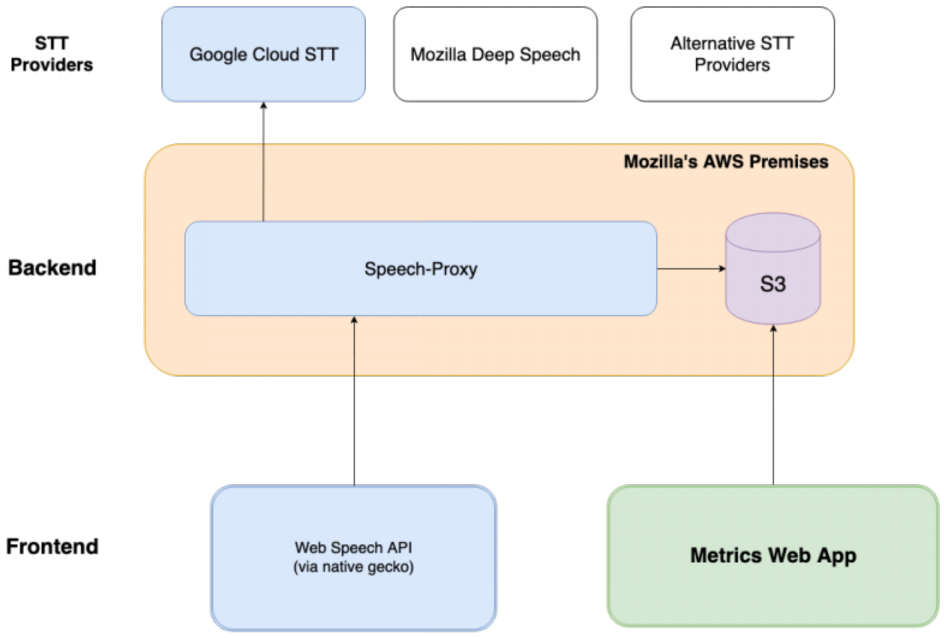


Figure 5.10: *Mozilla Speech-To-Text Cloud Architecture [29]*

Chapter 6

Results Analysis

After we described important details of implementation in the previous chapter, we center our attention in the evaluation and discussion of *Hand/Eye Actions*, as well as some *MES* scenarios where this project will be applied.

6.1 Trends

The trends are an assumed development in the future that will have a long-term and lasting effect on and change something. They are an important factor that can guide us about what people like or whether we should invest in any specific technology. If we look for historical data and combining the knowledge with other environmental factors, we will know what is happening now, and what is expected to happen tomorrow [7].

Analysing the trend regarding the past year of usage of *Eye Tracking* (figure 6.1) we see that in future the *Eye Tracking* investment could grow, but the certainty level is not very high. The data points are inconsistent, that means that our average growth rate is high, but the underlying numbers show alternating months of negative, flat and positive growth.

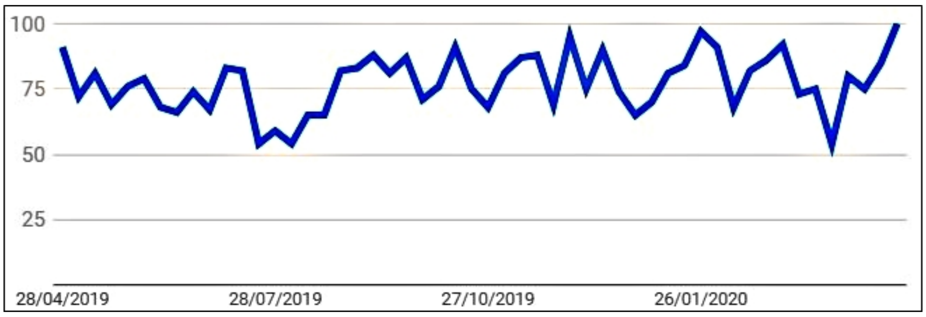
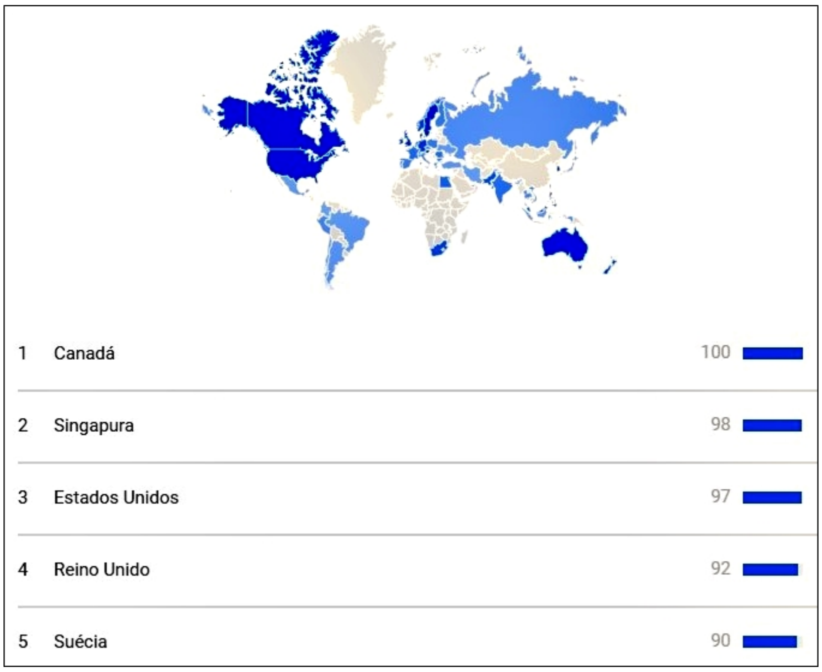


Figure 6.1: *Eye Tracking* Trends [38]

As *Michael Seibel* said, *ever-growing numbers of happy, loyal, and ideally paying customers*, is a strong indicator that you've found product-market fit. It also suggests that we understand the inner workings of our business, enough to know what levers to press to grow somewhat predictably over time.

The good news is the fact that the cities that use the most *Eye Tracking* are also the cities that produce the most semi-conductors or electronic components in the world (figure 6.2).



1	Canadá	100
2	Singapura	98
3	Estados Unidos	97
4	Reino Unido	92
5	Suécia	90

Figure 6.2: *Eye Tracking* Trends - World Map

6.2 Actions Evaluation

Evaluation metrics are a set of formulas that represent something that we want to measure, in order to evaluate a given process.

Evaluation metrics are specific to the type of machine learning task that a model performs. In this case the model is evaluated by measuring how well a predicted category matches the actual category.

The selected metrics for actions evaluation in real *MES* context are:

- **Duration** - Measures how long the algorithm takes to process the action;
- **Success Rate** - Success rate of detected actions;
- **Error Rate** - Failure rate of detected actions;
- **Actions Performed** - Total number of actions performed.

6.2.1 Hand Actions Evaluation

The *Hand Actions* conception was performed as a part of an iterative refinement process. In the beginning, *Hand Actions* had the following pattern movements:

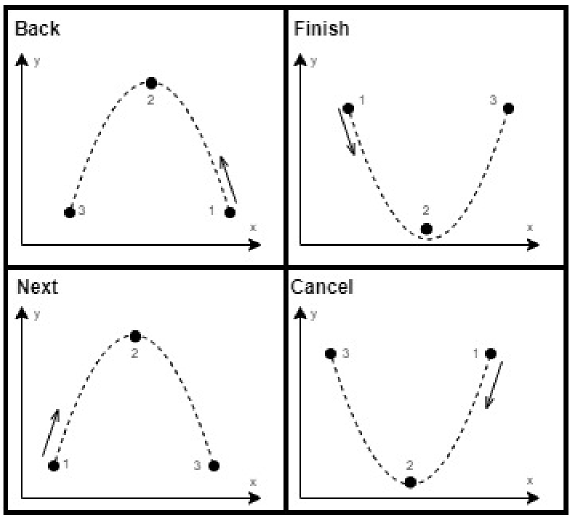


Figure 6.3: *Hand Actions* (version 1)

After testing the movements in a real *MES* context, it was noticed that sometimes the movement was not perfectly detected, and the complexity in terms of user experience was not the best. So it was decided to train the machine learning engine with simple and constant movements, as we can see in figure 6.4.

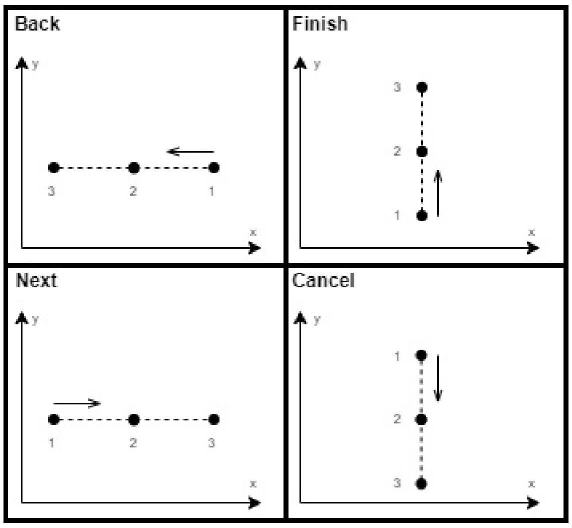


Figure 6.4: *Hand Actions* (version 2)

The *Hand Action* is a *Multi Classification* problem, and the algorithms associated with this type of problem that were tested are:

Averaged Perceptron, Fast Forest, Fast Tree, Lbfgs Logistic Regression, Lbfgs Maximum Entropy, Light Gbm, Linear Support Vector Machines, Sdca Maximum Entropy, Sgd Calibrated, Symbolic Sgd Logistic Regression.

The evaluation metrics chosen for the model training were:

- **Duration** - Measures how long the algorithm takes to run;
- **Macro Accuracy** - Computes the metric independently for each algorithm and then take the average;
- **Micro Accuracy** - Aggregates the contributions of all algorithms to compute the average metric.

In a multi-class classification task, micro-accuracy is preferable over macro-accuracy if we suspect there might be class imbalance. So, for choosing the best algorithm, was decided to use micro-accuracy as metric.

Regarding the training model, in the first scenario a dataset containing 120 gestures was used, grouped into 4 different actions (*Back, Next, Finish, Cancel*), where each action corresponds to 30 gestures.

After training the model for 10 seconds, the best results obtained were for the *Averaged Perceptron* algorithm with a 92,86 % of micro-accuracy (table 6.1).

Table 6.1: Models explored (10 seconds & 120 gestures)

Algorithm	**Micro Accuracy**	**Macro Accuracy**	**Duration**
AveragedPerceptronOva	0,9286	0,9500	1,0
SdcaMaximumEntropyMulti	0,9286	0,9500	2,5
LightGbmMulti	0,9286	0,9375	1,3
SymbolicSgdLogisticRegressionOva	0,2308	0,2500	1,0
Total experiment time	5,8840 s		

Changing training time to 200 seconds, the results are similar, with 92,86 % of micro-accuracy using *Stochastic Dual Coordinate Ascent (SDCA)* algorithm (table 6.2).

Table 6.2: Top 5 models explored (200 seconds & 120 gestures)

Algorithm	**Micro Accuracy**	**Macro Accuracy**	**Duration**
SdcaMaximumEntropyMulti	0,9286	0,9500	2,5
LightGbmMulti	0,9286	0,9375	1,3
LightGbmMulti	0,9286	0,9500	0,8
LightGbmMulti	0,9286	0,9583	1,4
SdcaMaximumEntropyMulti	0,9286	0,9500	20,0
Total experiment time	199,6917 s		
Total number of models explored	54		

If we reduce the size of training dataset to 80 gestures, where each action corresponds to 20 gestures, and we train the model for 200 seconds, the micro-accuracy decreases to 88.89 %, where the best algorithm selected is *Light Gradient Boosting Machine (LightGBM)* (table 6.3).

Table 6.3: Top 5 models explored (200 seconds & 80 gestures)

Algorithm	**Micro Accuracy**	**Macro Accuracy**	**Duration**
SdcaMaximumEntropyMulti	0,8889	0,9167	3,3
LightGbmMulti	0,8889	0,9167	0,8
FastTreeOva	0,8889	0,9167	7,9
LinearSvmOva	0,8889	0,9167	0,6
FastForestOva	0,8889	0,9167	8,0
Total experiment time	184,3570 s		
Total number of models explored	53		

The results of testing *Hand Actions* in real *MES* context are:

Table 6.4: *Hand Actions* in real *MES* context

Action	**Duration (s)**	**Actions Performed (un)**	**Success Rate (%)**	**Error Rate (%)**
Back	126,22	10	90,00	10,00
Next	152,96	10	80,00	20,00
Finish	170,72	10	80,00	20,00
Cancel	129,21	10	100,00	0,00

Image 6.5 shows another view of *Hand Actions* in real *MES* context, comparing only the success and error rates (for more details you can check the appendix A.1).

As we can see, for all actions the success rate is over 80%.

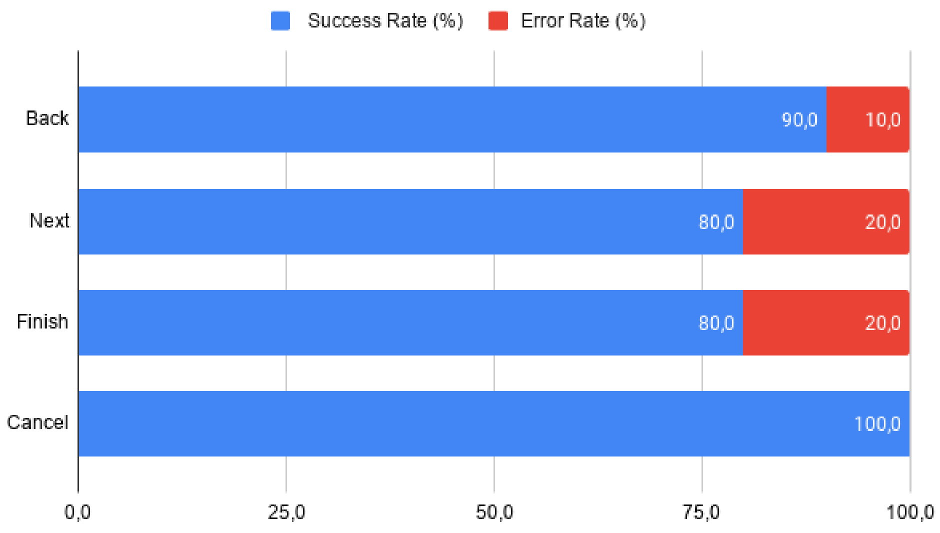


Figure 6.5: *Hand Actions* in real *MES* context

Since *Hand Actions* are part of a critical system, it's important to validate the time (table 6.5).

Table 6.5: *Hand Actions* in real *MES* context - Time Comparison

Action	**Total Duration (s)**	**Avg. Duration (s)**	**Max. Duration (s)**	**Min. Duration (s)**
Back	126,22	12,62	13,83	11,66
Next	152,96	15,30	25,03	12,31
Finish	170,72	17,07	33,12	12,08
Cancel	129,21	12,92	13,67	11,80

Total Duration	579,11 s
Avg. Duration	14,48 s
Max. Duration	33,12 s (Action Finish)
Min. Duration	11,66 s (Action Back)

From the moment we start the action with the movement of the hand until we get the *Hand Action* associated with that action triggering an action in the *MES*, it takes 14,48 seconds on average. However in some cases, due to external factors such as bad light conditions, it may take a little while to perform the action, which will not exceed the 33,12 seconds at the most (for more details, please see the appendix A.1).

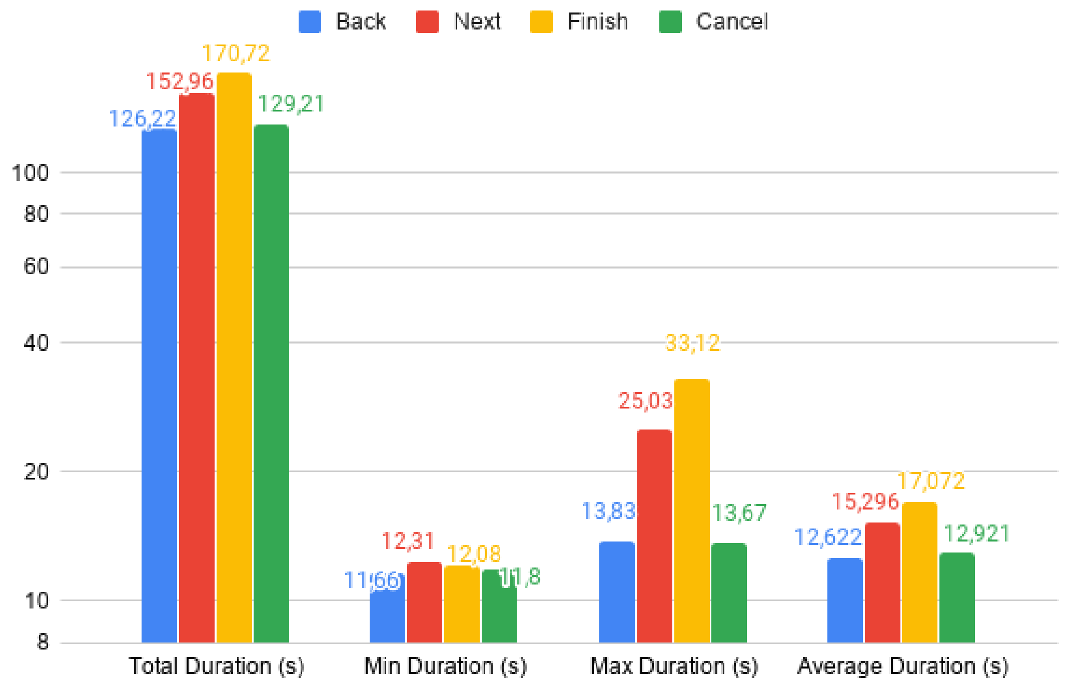


Figure 6.6: *Hand Actions* in real *MES* context - Time Comparison

Regarding *Hand Actions Evaluation*, we can conclude that time is not the factor that will increase the accuracy, but the size of the training dataset. So weighing all factors, the chosen algorithm was *Averaged Perceptron* with a 92,86 % of micro-accuracy. An additional important fact is the time that we take to perform an *Hand Action*, which will not exceed 33,12 seconds at the most and on averages it takes 14,48 seconds.

6.2.2 Eye Actions Evaluation

As mentioned in previous chapters, the *Eye Actions* are only used in finishing operations inside *Wizards* and *Execution Views*, and its correspondent patterns are represented in image 6.7.

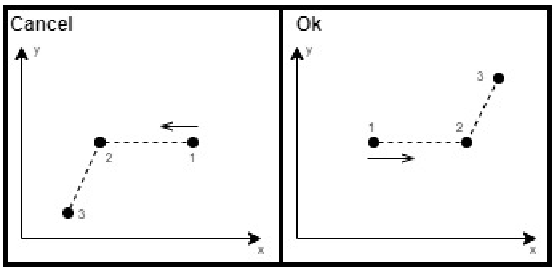


Figure 6.7: *Eye Actions*

The problem of *Eye Actions* is associated to *Binary Classification*, because we only have two possible states, the *OK* action or the *Cancel* action.

Binary Classification includes some popular algorithms like *Logistic Regression*, *k-Nearest Neighbors*, *Decision Trees*, *Support Vector Machine (SVM)*, and *Naive Bayes*. For this evaluation we only compare *Logistic Regression* and *SVM*, because they are the only algorithms that natively support two classes.

As an example: n = number of features; m = number of training examples.

- If n is large (1–10.000) and m is small (10–1000), we should use *Logistic Regression* or *SVM* with a linear kernel;
- If n is small (1–1000) and m is intermediate (10–10.000), we should use *SVM* with (Gaussian, polynomial etc) kernel;
- If n is small (1–1000), m is large (50.000–1.000.000+), first we manually add more features and then use *Logistic Regression* or *SVM* with a linear kernel.

Since we have only two simple actions and a small dataset, the chosen algorithm for *Eye Actions* is *Linear SVM*.

The evaluation metrics chosen for evaluate the model are:

- **Precision** - The number of positive/negative class predictions that actually belong to the positive/negative class;
- **Recall** - The number of positive/negative class predictions made out of all positive/negative examples in the dataset;
- **F-Measure** - Provides a single score that balances both the concerns of precision and recall in one number;

- **AUC** - Means area under the curve;
- **Accuracy** - The percentage of the correct classifications with respect to the all samples.

Testing *Eye Actions* with different datasets have the results shown in table 6.6.
In terms of accuracy we have approximately 50%:

- If $X_f - X_i$ is positive/negative and $Y_f - Y_i$ is positive/negative for *Cancel* action;

  And

- If $X_f - X_i$ is positive/negative points and $Y_f - Y_i$ is positive/negative for *Ok* action;

Table 6.6: *Eye Actions* Evaluation

Metrics	**100 Cases**	**1000 Cases**	**100 Cases** ($X_f - X_i$ & $Y_f - Y_i$ correct sign)	**1000 Cases** ($X_f - X_i$ & $Y_f - Y_i$ correct sign)
Accuracy	0,4958	0,4996	1,0000	1,0000
AUC	0,5059	0,4996	1,0000	1,0000
F1 Score	0,3648	0,3597	1,0000	1,0000
Negative Precision	0,4868	0,4995	1,0000	1,0000
Negative Recall	0,7227	0,7179	1,0000	1,0000
Positive Precision	0,5182	0,5000	1,0000	1,0000
Positive Recall	0,2818	0,2816	1,0000	1,0000

We always have 100% of accuracy independently of dataset:

- If $X_f - X_i$ is positive and $Y_f - Y_i$ is negative for *Cancel* action;

  And

- If $X_f - X_i$ is negative and $Y_f - Y_i$ is positive for *Ok* action;

The results of testing *Eye Actions* in real *MES* context are:

Table 6.7: *Eye Actions* in real *MES* context

Action	**Duration (s)**	**Actions Performed (un)**	**Success Rate (%)**	**Error Rate (%)**
Cancel	31,07	10	90,00	10,00
Ok	36,35	10	100,00	0,00

Image 6.8 shows another view of *Eye Actions* in a real *MES* context, comparing only the success and error rates (for more details, please see the appendix A.2).

As we can see, for all actions the success rate is over 90%.

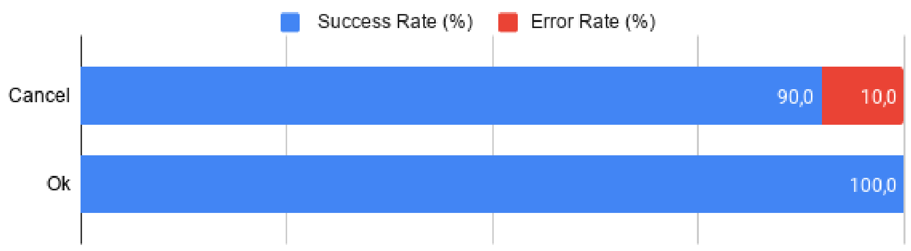


Figure 6.8: *Eye Actions* in real *MES* context

Since *Eye Actions* are part of a critical system, it's important to validate the time. The measurements are presented on table 6.8.

Table 6.8: *Eye Actions* in real *MES* context - Time Comparison

Action	**Total Duration (s)**	**Avg. Duration (s)**	**Max. Duration (s)**	**Min. Duration (s)**
Cancel	31,07	3,11	4,92	2,26
Ok	36,35	3,64	7,69	2,08

Total Duration	67,42 s
Avg. Duration	3,37 s
Max. Duration	7,69 s (Action Ok)
Min. Duration	2,08 s (Action Ok)

From the moment that we start to look for the central point and we perform the eyes movement until we get the associated *Eye Action* triggering an action in *MES*, takes on average 3,37 seconds. This average time seems very good. However in some cases, due to external factors such as a high latency of the *Message Bus*, it may take a little longer to perform the action, but that does not exceed 7,69 seconds at the most (for more details, please see the appendix A.2).

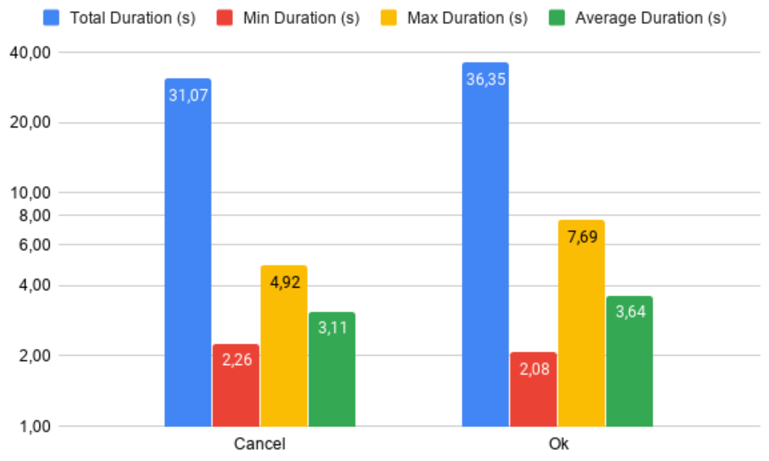


Figure 6.9: *Eye Actions* in real *MES* context - Time Comparison

Using *Eye Actions* proved to be a great feature, not only for the quick response of 3,37 seconds on average and for their accuracy, but also for being an asset when it comes to using extra validations to perform an action in the *MES* and conclude a click or selection event.

6.3 MES Scenarios

For further validation testing on *Business Data* objects like *Multimedia*, *Document* and *Site*, using two *MES* Scenarios with different types of complexity were also performed.

Scenario 1 : View an associated file with a *Multimedia* object.
Associated Steps:

1. *Eye Tracker Click* on *Business Data* sidebar menu item;
2. *Eye Tracker Click* on *Multimedia* tile, in the central panel;
3. In the central panel perform an *Eye Tracker Click* on *Multimedia* search input field and speak the name *"media"*;
4. *Eye Tracker Click* on *Multimedia* object searched;
5. *Eye Tracker Click* on *View* button, inside *Multimedia* page;

Scenario 2 : Edit one *Document* field and Create a *Site* object.
Associated Steps:

Edit Document

1. *Eye Tracker Click* on *Business Data* sidebar menu item;
2. *Eye Tracker Click* on *Document* tile, in the central panel;
3. In the central panel perform an *Eye Tracker Click* on *Document* search input field and speak the name *"document"*;
4. *Eye Tracker Click* on *Document* object searched;
5. *Eye Tracker Click* on *Edit* button, inside *Document* page;
6. Perform the *Cancel Hand Gesture*;
7. *Eye Tracker Click* on *Edit* button, inside *Document* page;
8. Inside *Wizard Edit Document* perform an *Eye Tracker Click* on *Description* field and speak the name *"example"*;
9. Inside *Wizard Edit Document*, perform an *Eye Tracker Click* on *Finishing* button and execute the *Eye Action Ok*;

Create Site

10. *Eye Tracker Click* on *Business Data* sidebar menu item;
11. *Eye Tracker Click* on *Site* tile, in the central panel;
12. *Eye Tracker Click* on *New* button, in the central panel;
13. Inside *Wizard Create Site* perform an *Eye Tracker Click* on *Name* field and speak the name *"porto"*;
14. *Eye Tracker Click* on *Next* button, inside *Wizard Create Site*;
15. Perform the *Back Hand Gesture*;
16. Inside *Wizard Create Site* perform an *Eye Tracker Click* on *Description* field and speak the name *"local"*;
17. Perform the *Next Hand Gesture*;
18. Perform the *Finish Hand Gesture*;

For each scenario the *Duration* and *Success Rate* were measured. The *Success Rate* is based in four factors:

- **Duration** - The time spent to perform the scenario (accounts for 10% of the weight defining the *Success Rate*);
- **Scenario Completion** - Successful execution of all steps (30% of weight);
- **Usability** - Refers to the ease of access and/or use of the functionalities (30% of weight);
- **Accessibility** - Refers to the design of products, devices, services, or environments so as to be usable by people with disabilities (30% of weight).

All of the factors associated to *Success Rate* have the same weight except *Duration*, because our main goal is to measure a solution that gives to operators a simple way to interact with software manufacturing systems without using their hands, what is implicitly related to *Scenario Completion*, *Usability* and *Accessibility*.

In table 6.9 we can see a comparison between scenarios (for more details, please see the appendix A.3).

The scenario type *one arm from screen* represents the normal posture that we have when sitting in front of a computer. The line of sight should be aligned with the top/center of the screen at a distance of 45 cm to 65 cm (more or less than one arm).

The scenario type *put down objects in first step* simulates a typical scenario when we put some object on the table and take off the gloves before we start to do a *MES* operation.

Table 6.9: *Scenarios Comparison*

Scenario	**With Tracking Solution**	**Type**	**Duration (s)**	**Success Rate (%)**
Scenario 1		One arm from screen	8,28	97,40
Scenario 1	Yes	One arm from screen	13,05	97,80
Scenario 1		Put down objects in first step	13,57	80,00
Scenario 1	Yes	Put down objects in first step	13,05	97,80
Scenario 2		One arm from screen	25,38	96,78
Scenario 2	Yes	One arm from screen	95,15	95,11
Scenario 2		Put down objects in first step	30,67	81,78
Scenario 2	Yes	Put down objects in first step	95,15	95,11

In general, scenarios 1 and 2 summarize the features associated with this project.

As we can see the usage of *tracking solutions* had an increase on *Success Rate* in most scenarios, although in some cases the time spent will be slightly longer. However, 5,29 seconds were not spent on *Put down objects in first step* in some scenarios (this time is based on putting objects on the table and take off the gloves before we start the scenario).

Due to the time that was spent in steps that has *Eye/Hand Actions* associated, it remains to be stated that the accuracy is quite good, like it was said in sections before.

Another important conclusion that was discovered is the fact that is not possible to use the tracking solution if we have more than one arm of distance to the screen and the microphone too far.

Figure 6.10 shows another summarized view of the scenarios performed.

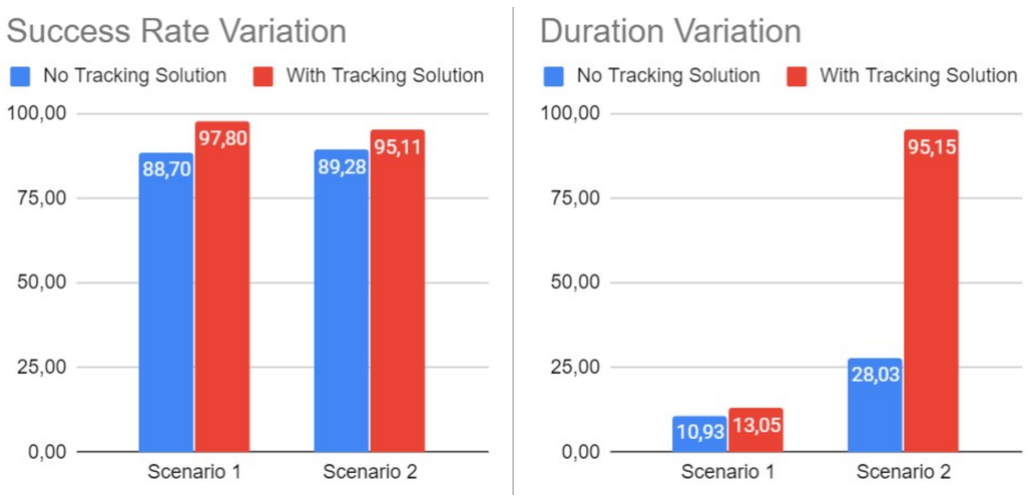
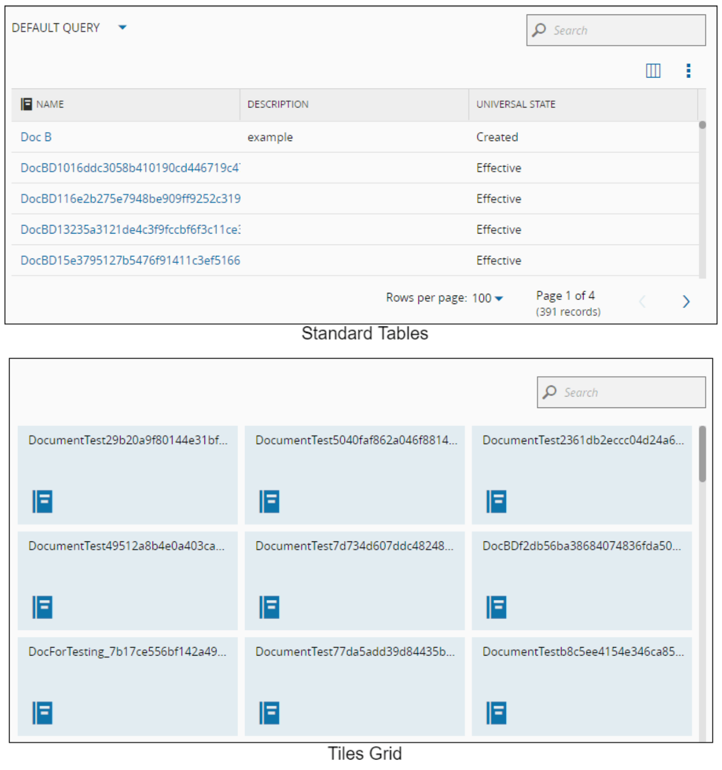


Figure 6.10: *MES* Scenarios - Comparison

The *Success Rate* was clearly increased, although it still could be improved, like in voice recognition that sometimes doesn't recognize our words. In another cases, *GUI* should be improved to have a better performance, like the case of multiple render editors, and when we have a *Wizard/Execution View*. In this last case the *DOM* elements that are in back planes of view, and because of its invisibility, could be destroyed instead of rendered.

Some *Standard Tables* should be also transformed into *Tiles Grid* (figure 6.11) so the user can easily perform an *Eye Tracker* click, with tracking solution.



NAME	DESCRIPTION	UNIVERSAL STATE
Doc B	example	Created
DocBD1016ddc3058b410190cd446719c4		Effective
DocBD116e2b275e7948be909ff9252c319		Effective
DocBD13235a3121de4c3f9fccbf6f3c11ce		Effective
DocBD15e3795127b5476f91411c3ef5166		Effective

Rows per page: 100 Page 1 of 4 (391 records)

DocumentTest29b20a9f80144e31bf...	DocumentTest5040faf862a046f8814...	DocumentTest2361db2eccc04d24a6...
DocumentTest49512a8b4e0a403ca...	DocumentTest7d734d607ddc48248...	DocBDf2db56ba38684074836fda50...
DocForTesting_7b17ce556bf142a49...	DocumentTest77da5add39d84435b...	DocumentTestb8c5ee4154e346ca85...

Figure 6.11: *Standard Tables* and *Tiles Grid*

The most important statement is that user can fully use the *MES* without using mouse or keyboard.

Chapter 7

Conclusions and Future Work

7.1 Main Contributions

The usage of traditional webcams for tracking the eyes has its difficulties, like knowing the exact position where the user is looking at. There are too many variables that can influence the result of *Eye Tracking*, like image quality, environment, lighting. If we want to build an efficient *Eye Tracking* solution, the better choice is to use a device that embeds an infrared light to illuminate the eye, causing highly visible reflections like the *Tobii Eye Tracker 4C*. The cameras of this platform, take high resolution images of the user eyes to be used in image processing algorithms and in physiological 3D models so we can estimate the position of the eye in space and the point of gaze, with high accuracy.

The usage of movements that have a pattern associated like *Hand/Eye Actions* to perform some operations has a great impact in user interaction and can be fully customizable. The *Eye Actions*, recognizing eye and iris movements, was proved to be a great feature, not only for the quick response of 3,37 seconds on average and for their accuracy, that is quite good, but also for being an asset when it comes to using extra validations to perform an action in the *MES*. If we use *Hand Actions*, the associated algorithm takes 14,48 seconds since the user performs the action until the algorithm returns the result (with 92,86 % of accuracy). However in some cases, due to external factors such as high *Message Bus* latency, it may take a little longer to perform the action.

In case of *MES* scenarios, we can efficiently use the developed *tracking solution*, since the *Success Rate* was increased compared to the current *MES* solution. The operators can perform parallel operations, like assemble and disassemble components, while interacting with the *MES*, without using mouse or keyboard.

The *MES* scenarios *Success Rate* could still be improved, focusing, for instance, on voice recognition, that sometimes don't recognize our words. In other cases, the *GUI* can be improved to have a better performance, like the case of multiple render editors, and when we have a *Wizard/Execution View*. In this last case the *DOM* elements that are in back planes of view, and because of its invisibly, could be destroyed instead of rendered.

The most important statement is that the user can fully use the *MES* without using mouse or keyboard, and customize the way he does this.

7.2 Future Work

I am sure this concept and project is the beginning of a new *MES* computer vision module. But there sill are many enhancements and features that can be added and developed, like:

- *Actions* complexity could be extended to use fingers and hand movements;
- *GUI* could be adaptable when we use a *tracking solution*, for more easily allow the user to perform the *MES* operations;
- Some *Standard Tables* should be also transformed into *Tiles Grid* allowing the user easily perform an *Eye Tracker* click, with the tracking solution;
- The voice recognition could use better algorithms;
- The *Eye/Hand Actions* could have a friendly way to be customized;
- Recording the images of the user eyes/hands when he use the tracking solution, could lead to a study in order to improve the *MES* functionalities.

Appendix A

Evaluation

The complete results for the *Hand Actions* and *Eye Actions* and for the two more complex scenarios described in chapter 6 are displayed in the next tables.

A.1 Hand Actions Evaluation

Tables A.1 and A.2 represents the detailed *Hand Actions* evaluation in real *MES* context.

Table A.1: Detailed *Hand Actions* Evaluation

Action	**Duration (s)**	**Status**	**Note**
Back	13,19	Fail	Lighting issues associated with hand detection.
	13,83	Success	
	13,26	Success	
	11,66	Success	
	13,09	Success	
	12,16	Success	
	11,78	Success	
	13,06	Success	
	12,25	Success	
	11,94	Success	

Table A.2: Detailed *Hand Actions* Evaluation (cont.)

Action	**Duration (s)**	**Status**	**Note**
Next	13,19	Fail	Lighting issues associated with hand detection.
	25,03	Success	
	13,95	Success	
	14,39	Success	
	19,34	Fail	Lighting issues associated with hand detection.
	13,46	Success	
	13,85	Success	
	14,51	Success	
	12,31	Success	
	12,93	Success	
Finish	13,25	Fail	Lighting issues associated with hand detection.
	33,12	Success	
	16,60	Fail	*Hand Action* fails.
	32,01	Success	
	12,08	Success	
	13,43	Success	
	12,65	Success	
	12,35	Success	
	12,64	Success	
	12,59	Success	
Cancel	13,67	Success	
	13,32	Success	
	13,14	Success	
	13,14	Success	
	13,64	Success	
	11,80	Success	
	12,88	Success	
	11,97	Success	
	12,43	Success	
	13,22	Success	

A.2 Eye Actions Evaluation

Table A.3 represents the detailed *Eye Actions* evaluation in real *MES* context.

Table A.3: Detailed *Eye Actions* Evaluation

Action	**Duration (s)**	**Status**	**Note**
Cancel	2,49	Success	
	4,92	Success	
	2,48	Success	
	3,29	Success	
	2,76	Success	
	2,35	Success	
	2,26	Success	
	3,90	Success	High latency of *Message Bus*.
	2,90	Success	
	3,72	Success	
Ok	3,15	Success	
	2,33	Success	
	2,08	Success	
	4,63	Success	High latency of *Message Bus*.
	3,03	Success	
	2,86	Success	
	4,53	Success	Too much time spent looking at the central point.
	2,65	Success	
	3,40	Success	
	7,69	Fail	High latency of *Message Bus*.

A.3 MES scenarios Evaluation

Tables A.4, A.5 and A.6 represent the detailed evaluation of *MES* scenarios 1 and 2. Each table line is composed by 7 parameters, where some of them have their specific calculation.

Time Rate is 100 % if the step time is less than 2,50 seconds, 90 % between 2,51 seconds and 5,00 seconds, 80 % between 5,01 seconds and 10,00 seconds, 70 % between 10,01 seconds and 20,00 seconds, 60 % between 20,01 seconds and 40,00 seconds and 50 % for more than 40,01 seconds.

Completion is 100 % if we can successfully complete the step and is 0 % if we can not complete the step.

Usability an *Accessibility* is 100 % if there is no failures, 90 % for 1 failure, 80 % for 2 failure, 70 % for 3 failure, 60 % for 4 failures and 50 % for more than 4 failures.

Table A.4: Scenario 1: One arm from screen

Step	Duration (s)	Time Rate (%)	Completion (%)	Usability (%)	Accessibility (%)	Note
Scenario 1: One arm from screen						
1	1,39	100	100	100	100	
2	0,99	100	100	100	100	
3	3,74	90	100	80	80	*GUI* rendering could be better. Input fields could be larger.
4	1,11	100	100	100	100	
5	1,05	100	100	100	100	
	8,28	**98**	**100**	**96**	**96**	
Scenario 1: One arm from screen (with tracking solution)						
1	2,16	100	100	100	100	
2	2,03	100	100	100	100	
3	5,39	80	100	70	100	*Voice Recognition* could be better.
4	1,57	100	100	100	100	
5	1,9	100	100	100	100	
	13,05	**96**	**100**	**94**	**100**	

Table A.5: Scenario 2: One arm from screen

Step	**Duration (s)**	**Time Rate (%)**	**Completion (%)**	**Usability (%)**	**Accessibility (%)**	**Note**
1	0,53	100	100	100	100	
2	0,59	100	100	100	100	
3	2,74	90	100	80	80	Bad *GUI* performance.
4	0,85	100	100	100	100	
5	1,18	100	100	100	100	
6	2,88	90	100	100	100	
7	1,11	100	100	100	100	
8	2,87	90	100	80	80	
9	0,85	100	100	100	100	
10	0,78	100	100	100	100	
11	0,92	100	100	100	100	
12	1,38	100	100	100	100	
13	2,03	100	100	70	70	Bad *GUI* performance with multiple editors.
14	1,44	100	100	100	100	
15	0,59	100	100	100	100	
16	3,26	90	100	80	80	Bad *GUI* performance with multiple editors.
17	1,05	100	100	100	100	
18	0,33	100	100	100	100	
	25,38	**97,78**	**100**	**95,00**	**95,00**	

Table A.6: Scenario 2: One arm from screen (with tracking solution)

Step	**Duration (s)**	**Time Rate (%)**	**Completion (%)**	**Usability (%)**	**Accessibility (%)**	**Note**
1	2,23	100	100	100	100	
2	2,55	90	100	100	100	
3	5,83	80	100	70	100	*Voice Recognition* could be better.
4	2,75	90	100	100	100	
5	2,31	100	100	100	100	
6	7,98	80	100	80	100	*Hand Action* could be faster.
7	2,69	90	100	100	100	
8	6,39	80	100	70	100	*Voice Recognition* could be better.
9	5,95	80	100	100	100	
10	2,13	100	100	100	100	
11	1,9	100	100	100	100	
12	2,48	100	100	100	100	
13	4,55	90	100	80	70	Bad *GUI* performance with multiple editors.
14	3,32	90	100	100	100	
15	9,35	80	100	80	100	*Hand Action* could be faster.
16	4,72	90	100	70	100	*Voice Recognition* could be better.
17	14,13	70	100	80	100	High latency of *Message Bus*. *Hand Action* could be faster.
18	13,45	70	100	80	100	*Hand Action* could be faster.
	95,15	**87,78**	**100**	**89,44**	**98,33**	

References

[1] V. Cehan R. G. Lupu A. Păsărică, R. G. Bozomitu. Pupil detection algorithms for eye tracking applications. *2015 IEEE 21st International Symposium for Design and Technology in Electronic Packaging (SIITME)*, October 2015.

[2] Peter J. Bex. Guido Maiello Agostino Gibaldi, Mauricio Vanegas. Evaluation of the tobii eyex eye tracking controller and matlab toolkit for research. *Behavior Research Methods*, 2016.

[3] Francisco Almada-Lobo. The industry 4.0 revolution and the future of manufacturing execution systems (mes). *Journal of Innovation Managements*, January 2016.

[4] Martin Arvidsson. Tools to deal with the shift to industry 4.0. https://www.tobiipro.com/blog/tools-deal-with-shift-to-industry-4-0/, March 2018.

[5] D. Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

[6] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[7] Barry Burns. *Trend Trading For Dummies*. 2014.

[8] Cuneyt Akinlar Cihan Topal, Halil Ibrahim Cakir. Appd: Adaptive and precise pupil boundary detection using entropy of contour gradients. *asd*, August 2018.

[9] Melvin E. Conway. *How do Committes Invent*. F. D. Thompson Publications, Inc., apr 1968.

[10] Derrick J. Parkhurst Dongheng Li. Starburst: A robust algorithm for video-based eye tracking. *Elsevier Science*, September 2005.

[11] Jason Babcock Dongheng Li and Derrick J. Parkhurst. Openeyes: A low-cost head-mounted eye-tracking solution. *Proceedings of the Eye Tracking Research & Application Symposium, ETRA 2006*, January 2006.

[12] David C. Van Essen and Charles H. Anderson. Information processing strategies and pathways in the primate visual system. *An Introduction to Neural and Electronic Networks, 2nd*, pages 45–76, 1995.

[13] David Fumo. Types of machine learning algorithms you should know. https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861, June 2017.

[14] P. Gupta. *Data Science with Jupyter: Master Data Science skills with easy-to-follow Python examples*. BPB Publications, 2019.

[15] Yoshua Bengio Ian Goodfellow and Aaron Courville. *Deep Learning (Adaptive Computation and Machine Learning series)*. The MIT Press (November 18, 2016), 2016.

[16] Ian Spence Ivar Jacobson and Kurt Bittner. *USE-CASE 2.0 - The Guide to Succeeding with Use Cases*. Ivar Jacobson International SA, dec 2011.

[17] Barati Morteza Walsh Vincent Tcheang Lili Javadi Amir-Homayoun, Hakimi Zahra. Set: a pupil detection method using sinusoidal approximation. *Frontiers in Neuroengineering*, April 2015.

[18] John R. Koza, Forrest H. Bennett, David Andre, and Martin A. Keane. *Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming*. Artificial Intelligence in Design '96, 1996.

[19] Philip A. Laplante. *Comprehensive Dictionary of Electrical Engineering*. CRC Press, 2005.

[20] Neil A. Dodgson Lech Swirski, Andreas Bulling. Robust real-time pupil tracking in highly off-axis images. *ETRA '12*, March 2012.

[21] David Leggett. A brief history of eye-tracking. https://www.uxbooth.com/articles/a-brief-history-of-eye-tracking/, January 2010.

[22] Barbara H. Liskov and Jeannette M. Wing. A behavioral notion of subtyping. *ACM TransactIons on Programmmg Languages and Systems, Vol 16, No 6*, nov 1994.

[23] Critical Manufacturing. Mes for industry 4.0. https://www.criticalmanufacturing.com/en/critical-manufacturing-mes/complete-modular-solution, 2019.

[24] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hal, sep 2017.

[25] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall, 1997.

[26] Microsoft. Linearsvmtrainer class (microsoft.ml.trainers). https://docs.microsoft.com/en-us/dotnet/api/microsoft.ml.trainers.linearsvmtrainer?view=ml-dotnet, 2020.

[27] Microsoft. Ml.net documentation. https://docs.microsoft.com/en-us/dotnet/machine-learning/, 2020.

[28] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. Adaptive computation and machine learning series. MIT Press, 2012.

[29] Mozilla. Web speech api - speech recognition. https://wiki.mozilla.org/Web_Speech_API_-_Speech_Recognition/, 2020.

[30] M. Jones P. Viola. Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, dec 2001.

[31] Bartosz Pietrucha. Angular architecture patterns and best practices. https://angular-academy.com/angular-architecture-best-practices/, jul 2019.

[32] Tobii Pro. Why do we move our eyes. https://www.tobiipro.com/learn-and-support/learn/eye-tracking-essentials/why-do-our-eyes-move/, aug 2020.

[33] Andrew G. Barto Richard S. Sutton. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning series) 2nd Edition*. A Bradford Book; second edition edition (November 13, 2018), 2018.

[34] F. Rosenblatt. *The perceptron: A probabilistic model for information storage and organization in the brain*. Psychological Review, 1958.

[35] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach 3rd Edition*. Pearson Education India; 3rd edition (October 14, 2015), 2015.

[36] Humberto Santos. História do mes - manufacturing execution system | flow. https://flowtech.pt/pt/blog/historia-mes-manufacturing-execution-system/, aug 2916.

[37] Tereza Soukupová and Jan Cech. Real-time eye blink detection using facial landmarks. *21st Computer Vision Winter Workshop*, February 2016.

[38] Google Trends. Google trends 2019. https://trends.google.pt/, 2019.

[39] Katrin Sippel Wolfgang Rosenstiel Wolfgang Fuhl, Thomas Kübler and Enkelejda Kasneci. Excuse: Robust pupil detection in real-world scenarios. *Springer LNCS 9256. Editor of the proceedings of the 16th International Conference*, September 2016.

[40] Thomas C. Kübler Enkelejda Kasneci Wolfgang Fuhl, Thiago Santini. Else: ellipse selection for robust pupil detection in real-world environments. *Ninth Biennial ACM Symposium*, March 2016.