UNIVERSITÀ DEGLI STUDI DI CATANIA

Dipartimento di Matematica e Informatica

Corso di Laurea Triennale in Informatica

Marco Ardizzone

Data streaming for real-time object-interaction recognition and analytics using HoloLens2

Relazione Progetto Finale

Relatore:
Antonino Furnari

Anno Accademico 2020 - 2021

Abstract

This thesis's goal is to create a system which is able to perform real time object detection and interaction recognition, returning to the user statistics about the usage of objects through an insightful dashboard. This system is devised for the purpose of increasing productivity in an industrial environment, since it detects interactions with work tools, it gives back some useful dashboards which could be analyzed for retrieving usage information. Another possible use case for the proposed system is to enhance job safety in a workplace, since the application detects interactions and attended at objects. The client system runs on **Hololens2** [1]: once per 2 seconds the smartglasses capture a photo from the user's point of view and collect location of hand joints and eye gaze. This data is then wrapped in a JSON object and sent via a **socket connection** to the server. The server runs on a **Docker** [9] VM located in a remote server on Ubuntu: the server receives the Json objects in **streaming**, stores them in a queue and analyzes them as soon as possible. Once an object is analyzed, data about user interaction is stored and presented in a dashboard. The system has been evaluated by comparing it to a pre-existing Research Mode [15] application, named StreamRecorder-App [19]. To assess the usefulness of the application, we performed a user study including 10 people, in order to evaluate its usability. The outcome of the test suggests that the system performs better when detectiong interacted objects rather than attended ones, that is probably due to the fact that the eye gaze tracking capabilities provided by Hololens2 are not always accurate. The proposed system is still a prototype, it has some issues and could be improved, for instance making the 3D to 2D mapping of gaze coordinates dynamic or improving low level code in order to make the system run more smoothly.

Contents

Chapter 1

Introduction

The goal of this thesis is to build a system able to monitor interactions with objects performed by a user wearing a Microsoft Hololens2 [1] device. The proposed system makes use of a data streaming and processing pipeline in order to transmit images and high-level information about hand joints and gaze from Hololens2 to a server, which in turn processes the data using object detection algorithms.
This chapter will introduce the problem, discuss the motivation, and give an outline of the thesis.

1.1 Problem Definition

Wearable technologies are widespread nowadays: using them it is possible to obtain lots of data in real time, such as geolocation data, images or videos, as well as gyroscope data.
Microsoft Hololens2 allows to capture photos or videos, track the position of hands and eyes and perform spatial mapping.
The goal of this thesis is to retrieve data about hand and eye tracking while Hololens2 is worn in an industrial environment, perform object detection in order to extract information about what kind of objects the user has been interacting with, then releasing some dashboard and infographics for the purpose of analyzing the behaviour of the user.
See *Figures 1.1, 1.2* for an example of the system output.

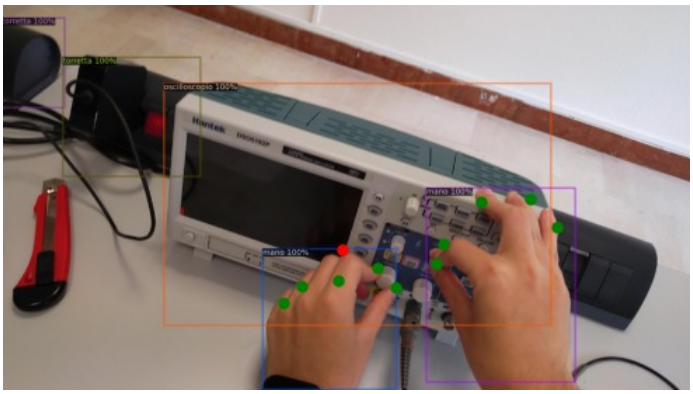


Figure 1.1: Example of the system's output.

Figure 1.2: Example of Kibana's dashboard.

1.2 Motivations

This project aims to analyze what happens in an industrial environment, which may be helpful in order to increase productivity and safety. Over the last few years Industry 4.0 [6] and Internet of Things [7] have been dealing with connected devices that can send and receive lots of data while they are worn. Among these solutions, wearables have shown a great potential in industrial environments, since they allow workers to complete their tasks having their hands free, while receiving feedback and collecting data. Despite this potential, wearable technology is not completely used in industrial environment, mostly because technical solutions are immature. This thesis proposes the prototype of a system which is capable of monitoring object interactions in industrial facilities, allowing to increase efficiency and productivity, as it is possible to monitor tools usage and its wear. Also, it is possible to increase safety and reduce industrial accidents (more than 2 million of people die at work each year [8]). Such systems (*See Figure 2.9*) may report whether an employee is not using well a working tool, or whether they are not looking at it while using it, which may imply a dangerous behavior.

1.3 Outline

In the next chapters, there will be an overview of which tools have been used in this thesis, along with a detailed solution proposed for the problem illustrated above and an evaluation of the given solution.

- Chapter 2: an overview of the tools used in this thesis.
- Chapter 3: the proposed solution is presented.
- Chapter 4: an evaluation of the proposed solution.
- Chapter 5: conclusion and possible enhancements.
- Chapter 6: an appendix containing details about the proposed solution.

Chapter 2

Tools and Algorithms used in this thesis

This chapter introduces every tools and algorithms used in this thesis.
For the purpose of this thesis, we have relied on a pair of **Hololens2** [1], the data source which will provide photos and high level information.
In order to access these data, Hololens2 has been programmed in *C#* on a **Unity** [4] and **MRTK-Unity** [5] environment.
The data are then embedded on a Json file and sent via a **socket connection** opened on heterogeneous languages and operating systems (C# on Hololens2 equipped with Windows and Python on a Linux server).
On the server, the application runs on a **Docker** [9] set of containers. These containers are connected in the following way: **Logstash** [10] reads the data coming from the socket and sends them to **Apache Kafka** [11] in a formatted way, continuously. Kafka saves these logs and send them as a stream of data to **Apache Spark** [12].
Spark is a data processing framework that can quickly perform processing tasks on very large data sets.It is used to perform **Detectron2** [2], a library that provides state-of-the-art detection and segmentation algorithms. Detectron2 is used to detect which objects are present in scene and these data are aggregated to the high level information coming from Hololens2, namely coordinates of hand tips joints and eye gaze, in order to give information about object interaction. Once the detection is done, a list of detected/interacted object is returned to **ElasticSearch** [13].
ElasticSearch is a distributed, open-source search and analytics engine, used for storing the output of the detection and serve it to **Kibana** [14].
Kibana is an open user interface which allows to visualize ElasticSearch data. It is used for query ElasticSearch data and present some insightful dashboards. The following sections discuss each of the used tools.

2.1 Hololens2

Microsoft Hololens2 [1] is a pair of smartglasses for Augmented Reality (*See Figure 2.1*): it has several cameras (up to a resolution of 2K) and sensors (*See Figures 2.2, 2.3*). It can detect Hands, Eye Gaze, Voice and can also perform spatial mapping. The Hololens2 device is fundamental in this thesis because it can collect a lot of data in real time, while the employee is using work tools and wearing it.

Figure 2.1: A pair of Microsoft Hololens2.

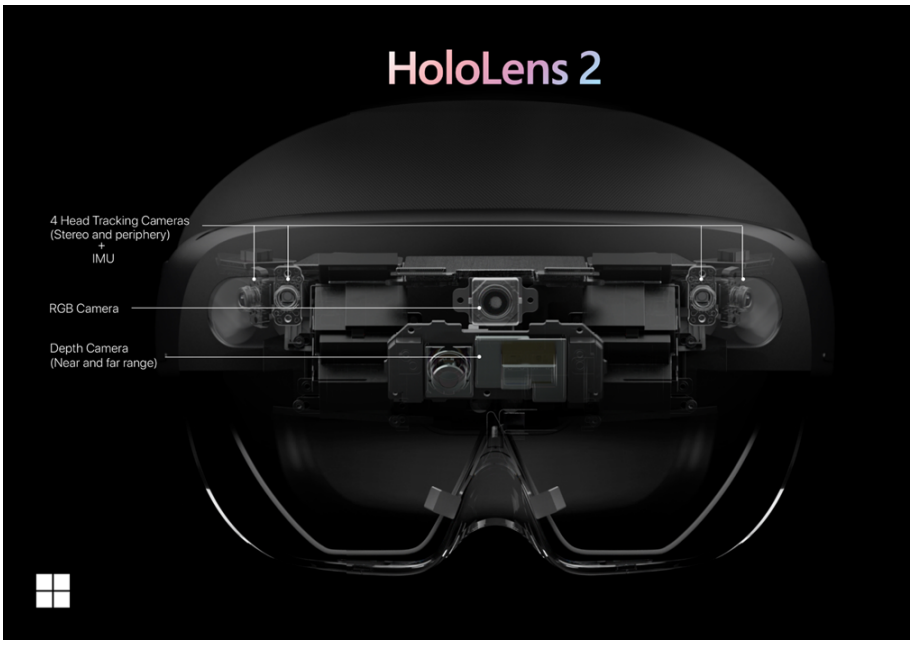


Figure 2.2: Microsoft Hololens2's frontal sensors.

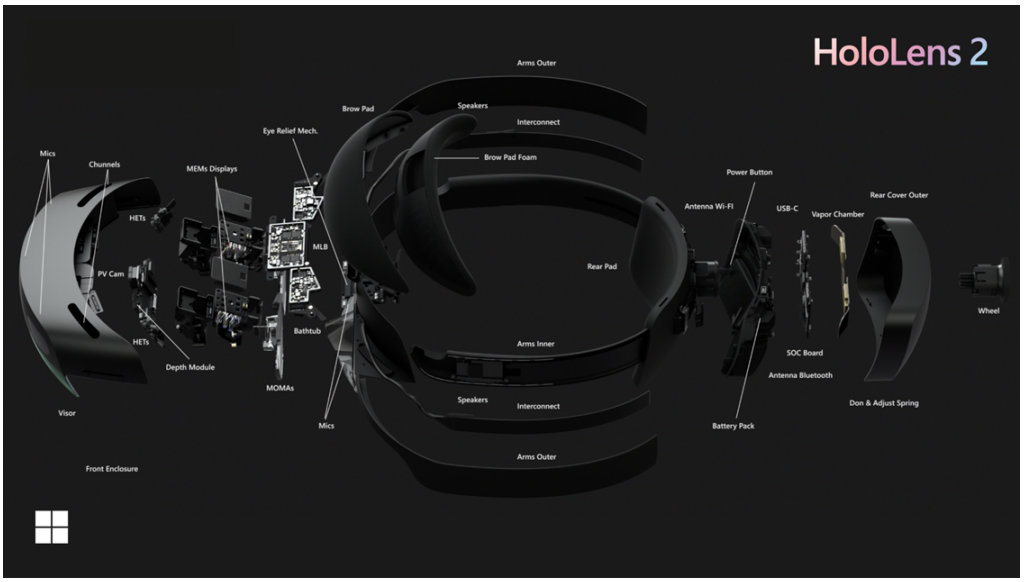


Figure 2.3: Microsoft Hololens2's structure.

2.2 Unity and MRTK-Unity

Hololens2's scripts have been written in *C#* language in a Unity [4] environment, using Mixed Reality ToolKit in Unity [5] (Example of MRTK Hub at *Figure 2.4*).
MRTK (*See Appendices .1, .6*) is a toolkit that provides a set of components and features used for developing AR applications. It has been employed in:

- **Hand tracking**: Hololens2 can detect whether a hand is present in the scene and eventually track it, since it can detect 25 different hand joints, such as finger tips, knuckles or palm. See *Figure 2.5*.
- **Eye gaze tracking**: Hololens2 has eye-tracking cameras that can estimate the gaze origin. This information is then combined with gaze direction, in order to give information about where the user is gazing at.
- **User Interface**: MRTK provides countless prefabs to speed up UI process. In this thesis we have used some buttons and a virtual keyboard. See *Figures 2.6, 2.7, 2.8*.
- **Web Browser**: It is implemented in asynchronous way, in order to open the output dashboard while the application is running and give some feedback to the user. See *Figure 2.9*

Figure 2.4: MRTK Hub.

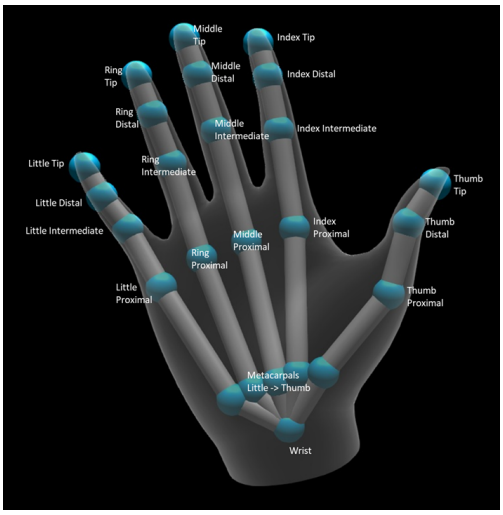


Figure 2.5: Hololens2 Hand tracking joints.

Figure 2.6: Enter IP input text.

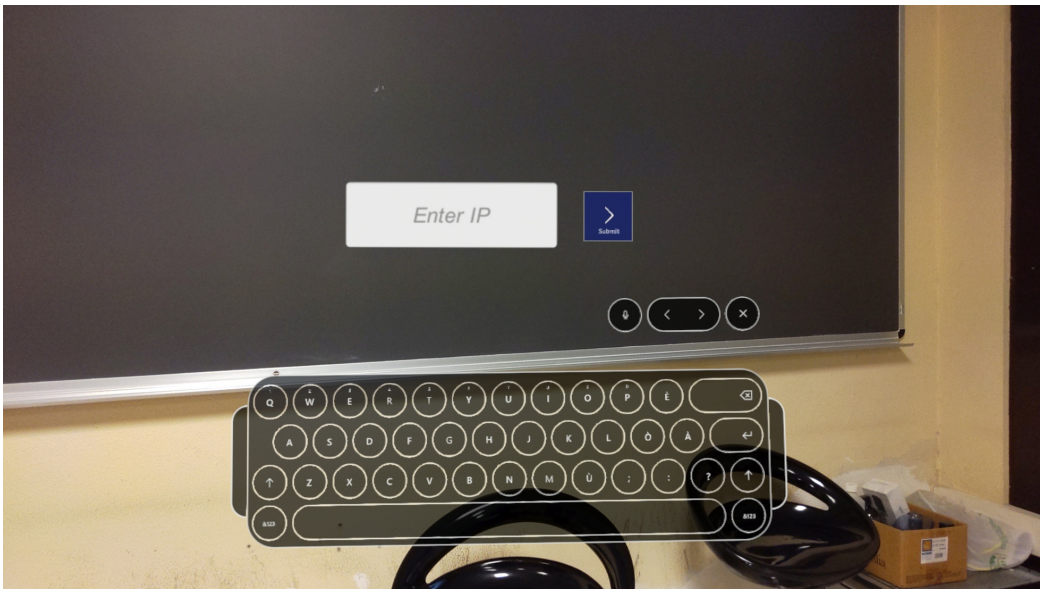


Figure 2.7: Virtual Keyboard.

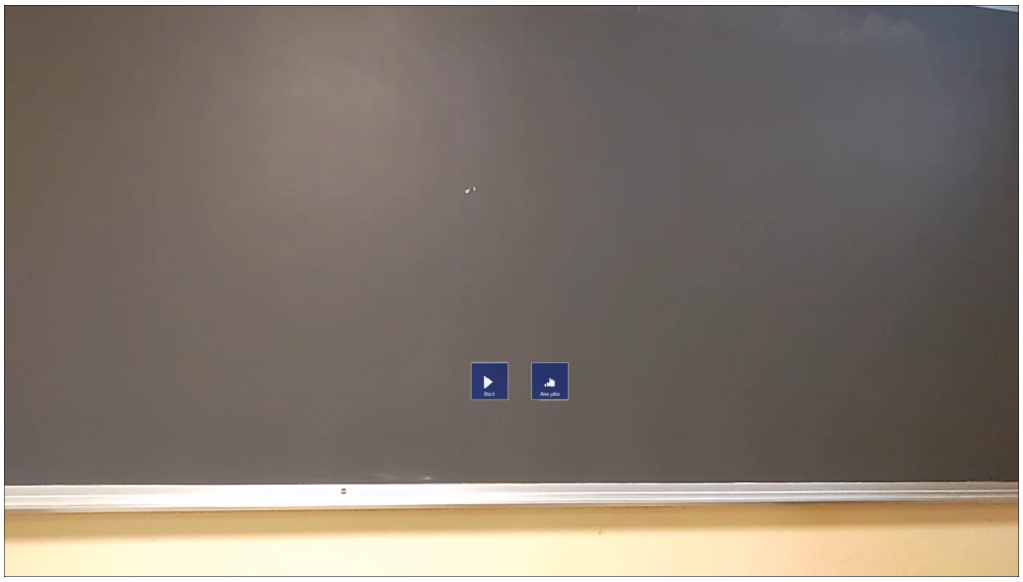


Figure 2.8: Start/Stop and Analytics buttons.

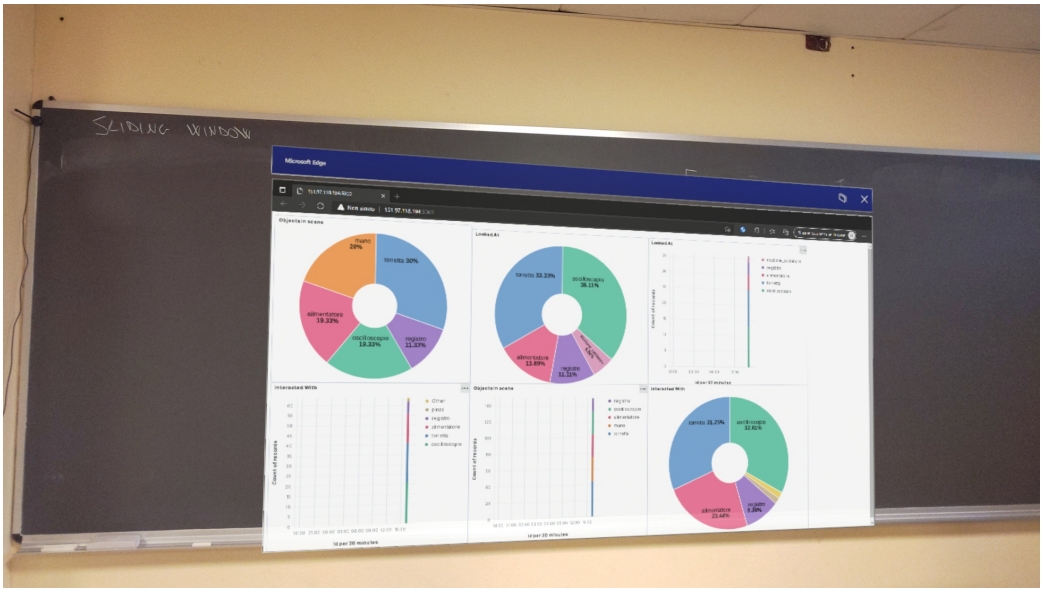


Figure 2.9: AR dashboards.

2.3 Detectron2

Written in *Python*, Detectron2 uses deep learning for object detection, providing an easy-to-use and extremely powerful framework for performing the detection of objects in photos or videos (*See Figure 2.10*). It is used in this project for getting information about what object is the employed interacting with (*See Appendix .2*). A custom model has been used for detection: instead of using COCO dataset, we have choosen to use a model trained on work tools such as *Oscilloscopes*, *Power Suppliers* and *Welders*. (*See Appendix .3).*

Object Detection Object detection is a technology related to Computer Vision [3] that deals with detecting instances of objects of a certain class in images or videos. It has many applications, such as face detection, video surveillance or pedestrian detection on automated guided vehicles. Object Detection is fundamental in this thesis because it is used to locate the objects in the scene. This information will be helpful for giving insights about object usage.

Figure 2.10: An example of Detectron2's output.

Faster R-CNN Detectron2 uses a Faster R-CNN model [26] to perform object detection (See structure at *Figure 2.11*).

Faster R-CNN works as follows:

- A Convolutional Neural Network [27] is applied to the input image in order to extract features.
- The feature maps are passed to the Region Propsal Network (RPN) [26], which will produce a series of candidate bounding boxes on present objects.
- Once bounding box are detected, a fixed-length vector representation is extracted for each of them using a RoI Pooling Layer [28].
- Finally, the detected objects are classified to assign an object a class.

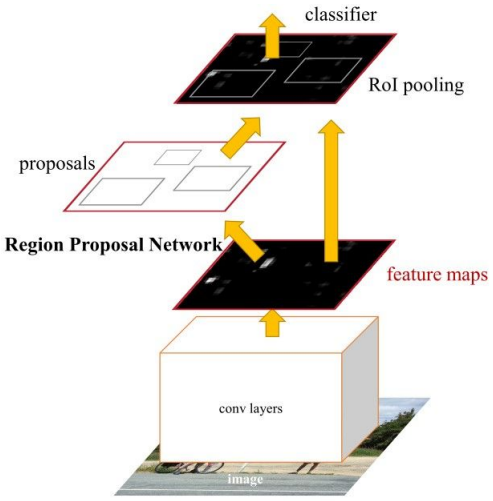


Figure 2.11: Faster R-CNN architecture.

2.4 Socket Connection

A socket is a communication channel, used for sending and receiving data across the network. This communication channel allows heterogeneous systems to communicate, since Hololens2 and the server are based on different operating systems. A client socket (*C#*) is opened in Hololens2 [1], on a Windows10 environment. The client socket sends logs about hand finger tips and eye gaze coordinates and a photo, once per 2 seconds.
A server socket (*Python*) is opened on a remote computer, on a Linux (Ubuntu 20.04) environment, whose receives and converts logs and photos from Hololens2 and delivers them to Logstash [10].

2.5 Docker

Docker [9] (See structure at *Figure 2.12*) is an open source containerization platform. It enables developers to package applications into containers—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment. Containers simplify the build, deployment and management of distributed application. The container will start the *Dockerfile*, a configuration file which contains references for downloading and running the application and its dependencies. Multiple containers can be run at the same time using a *docker-compose.yml* configuration file. The *docker-compose.yml* configuration file contains references to applications' *Dockerfiles* that will download and run all the dependencies, letting the software system to be portable.
Docker is used in this thesis for connecting every framework in the data pipeline and making it a single distributed application. As a result, the software system is portable and could easily run on any hardware configuration, regardless of whether is running on a laptop, desktop or server.

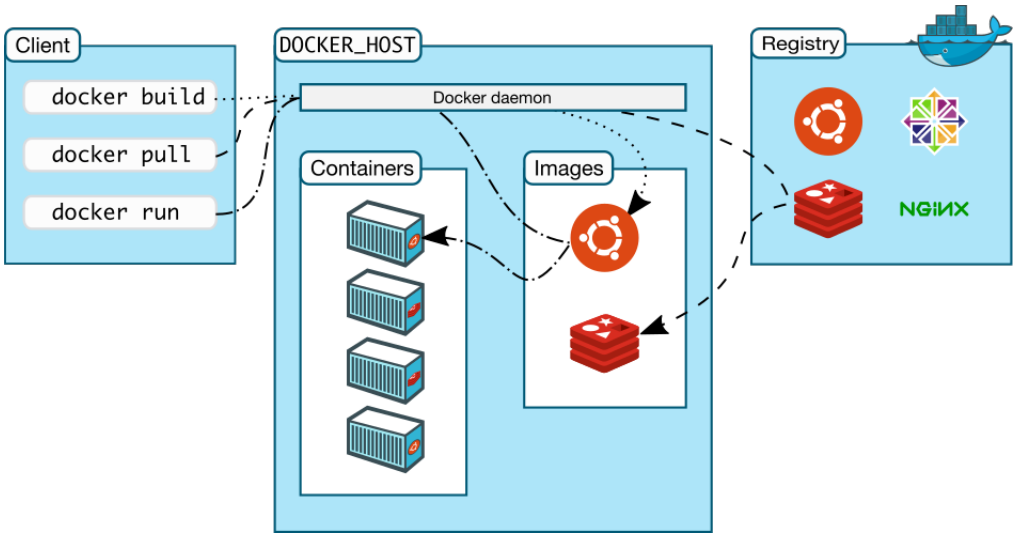


Figure 2.12: Docker's Architecture.

2.6 Logstash

Logstash (See structure at *Figure 2.13*) is an open source data collection engine with real-time pipelining capabilities. It can dynamically merge data from different sources and normalize them into a destination of your choice. It is used for getting raw data from the server socket and prepare them into log, then passing them to Kafka [11].

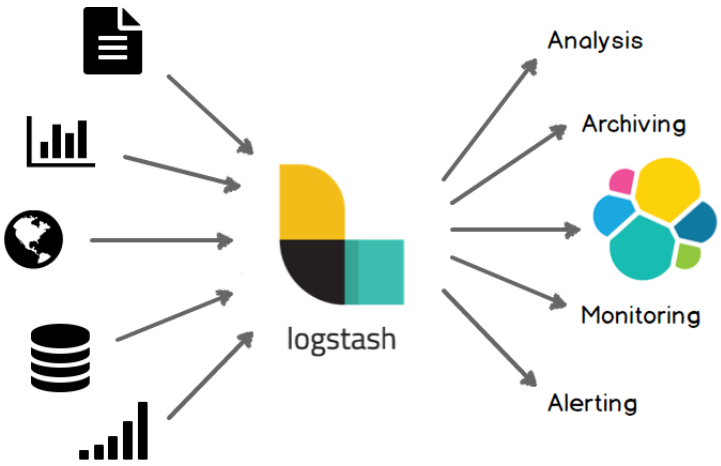


Figure 2.13: Logstash's Architecture.

2.7 Kafka

Apache Kafka (See structure at *Figure 2.14*) is an open-source software which provides a framework for storing, reading and analysing streaming data. It is used for handling real time data feeds with low-latency and high-throughput. It is used for receiving logs from Logstash, storing, and sending them as a stream of data to Spark [12], which will use Detectron2 for performing object detection.

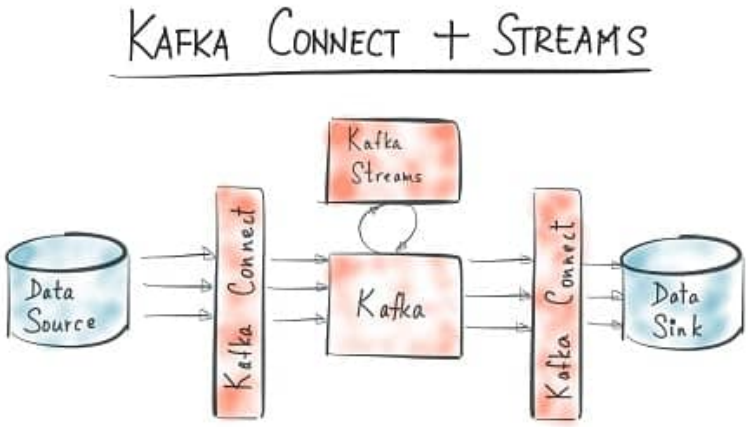


Figure 2.14: Kafka's Architecture.

2.8 Spark

Apache Spark (See structure at *Figure 2.15*) is an open-source unified analytics engine for large-scale data processing. It allows to program in *Java*, *R*, *Scala* and *Python*, ensuring parallelism and fault tolerance. It is used for applying object detection to the stream coming from Kafka. Once Detectron2 has analyzed the data, it returns to Elastic Search a list which contains objects that the user has been interacting with.

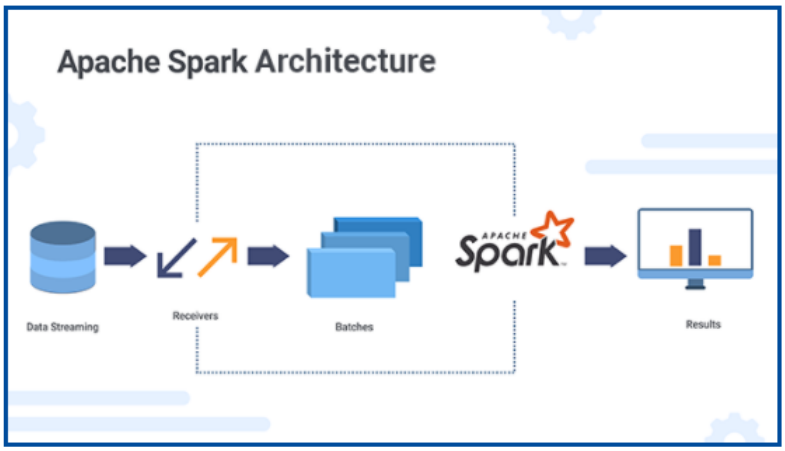


Figure 2.15: Spark's Architecture.

2.9 Elastic Search & Kibana

Elastic Search (See structure at *Figure 2.16*) is a distributed, free and open search and analytics engine for all types of data. It stores object detection information from Spark, allowing Kibana [14] to query them for doing data visualization. Kibana (See structure at *Figure 2.16*) is a data visualization and exploration tool used for performing time-series analytics, application monitoring, and operational intelligence use cases. It offers powerful and easy-to-use features such as histograms, line graphs, pie charts, heat maps, and built-in geospatial support. It is strongly integrated with Elastic Search, so it allows to easily visualize data stored there.

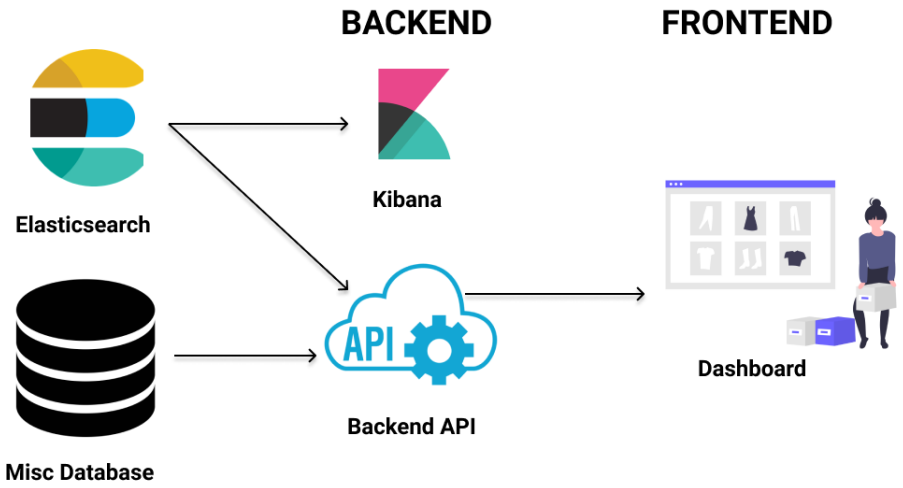


Figure 2.16: Elasticsearch and Kibana's usage.

Chapter 3

Proposed Solution

The problem considered in this thesis is to obtain and process a real time stream of data from Hololens2, in order to understand whether the user is interacting with an object or not. The proposed solution follows the scheme shown in *Figure 3.1*.

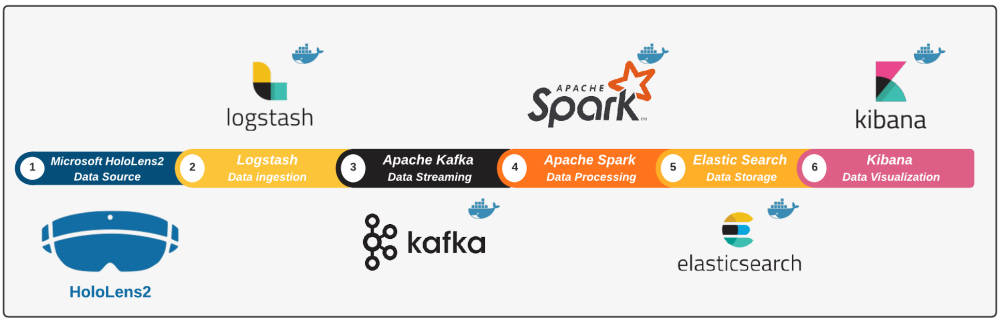


Figure 3.1: Data Pipeline.

1) Data Source A *C#* client socket on Hololens2 sends, once every 2 seconds, a Json file.

Figure 3.2: Socket connection between Hololens and Logstash.

The json file is structured in the following way:

- a formatted string which contains data about eye gaze and hand finger tips 3D Hololens2's coordinates and a timestamp (UNIX format), used as ID.
- an image captured by Hololens2's camera, encoded in *base64*. [16]

See *Listing 3.1* for Json file example.

```
1  {
2  "msg":"1634285036194 0.005289476 -0.4893723 0.07160407
       0.09180956 0.00176724 -0.4991095 -0.01308681 0.5108312
       0.04264754 0.04264754 -0.02333403 -0.005289476
       -0.01308684 0.01332481 0.00176724 -0.09180956
       0.005289476 0.01308684 -0.06470962 -0.07160407
       0.4893723 -0.04264754 0.005289476 0.4991095 -0.00176724
        -0.08928258 0.5108312 0.01308681 -0.09587268 0.4899464
        0.02333403 -0.09180956 0.4916802",
3  "img":"/9j/4AAQSkZJRgABAQ....//Z"
4  }
```

Listing 3.1: Json Example.

Meanwhile, a *Python* server socket, located on a remote computer, receives and decodes the images and the data logs from Hololens2's socket, converts coordinates from Hololens2's coordinate system to 2D pixel coordinates system and write 1 row on a shared *.csv* file. *See Appendix .5*

2) Data Ingestion Logstash reads from that *.csv* file and sends formatted logs to Kafka. *See Listing 3.2 for Logstash config file.*

```
input{
    file{
        path=>"/usr/share/logstash/csv/logs2D.csv"
        start_position=>"beginning"
    }
}

filter{
    csv{
        separator=>","
        columns=>["id", "c1", "c2", "c3", "c4", "c5", "c6",
        "c7", "c8", "c9", "c10", "c11", "c12", "c13", "c14",
        "c15", "c16", "c17", "c18", "c19", "c20", "c21", "c22"
            ]
    }
    mutate{
        convert => {
          "c1" => "integer"
          "c2" => "integer"
          "c3" => "integer"
          "c4" => "integer"
          "c5" => "integer"
          "c6" => "integer"
          "c7" => "integer"
          "c8" => "integer"
          "c9" => "integer"
          "c10" => "integer"
          "c11" => "integer"
          "c12" => "integer"
          "c13" => "integer"
          "c14" => "integer"
          "c15" => "integer"
          "c16" => "integer"
          "c17" => "integer"
          "c18" => "integer"
          "c19" => "integer"
          "c20" => "integer"
          "c21" => "integer"
          "c22" => "integer"
        }
        remove_field => ["@timestamp", "@version", "host", "
            path", "message"]
    }
}


```

```
output{
    kafka {
        codec => json
        topic_id => "tap"
        bootstrap_servers => "kafkaserver:9092"
    }
}
```

Listing 3.2: Logstash config file.

These logs are formatted as shown on *Table 3.1* and sent to Kafka in *json* format.

image id text/date	x1 int	y1 int	... int	... int	x21 int	y22 int

Table 3.1: Data logs.

We have 22 values, since every image has at most 11 interaction point (Eye Gaze, Left Hand tips and Right Hand tips).

3) Data Streaming Kafka gets logs from Logstash and acts as a queue, storing them until Spark can analyze them. Then sends a stream of data to Spark and wait for it to process it before sending another stream.

Figure 3.3: Kafka as a queue.

4) Data Processing Spark receives the data stream and does the heavy computation: it takes a photo and its logs, applies Detectron2 [2] and checks whether there is a known object nearby hand joints or eye gaze coordinates (*See Appendix .7*). In this case, an *interaction* is detected and a list of the interacted objects is sent to Elastic Search. See *Listing 3.3* for Spark's output example at *Figure 3.4*

```
{
 "id":"1635952242420",
 "Looked at": "[oscilloscopio]",
 "Interacted with": "[oscilloscopio, torretta]",
 "Object in scene": "[oscilloscopio, torretta]"
}
```

Listing 3.3: Spark's output.

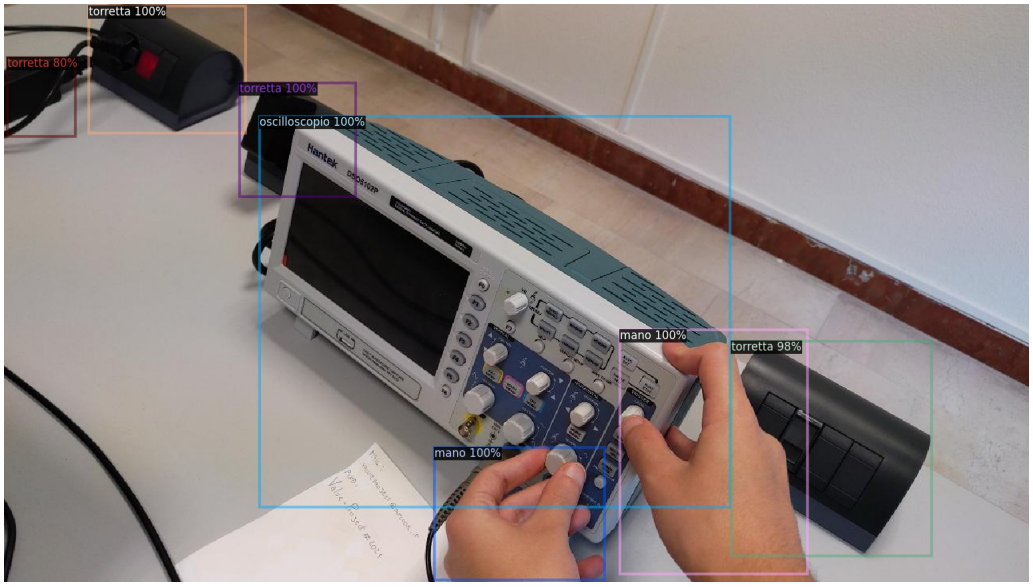


Figure 3.4: Example of detected image.

5) Data Storage Elastic Search receives data from Spark, stores them and allows Kibana to query them for doing data visualization. Data are stored in a format which can be seen in *Table 3.2*:

image id text/date	[Looked at objects] text	[Interacted with objects] text	[Objects present in scene] text

Table 3.2: Data entry.

6) Data Visualization Finally, Kibana receives data from Elastic Search and shows a dashboard, allowing to analyze which objects the user has been interacted with the most. An example of these dashboard can be seen in *Figure 3.5*:
These dashboards can be checked on the remote server but also on Hololens2 (*See Appendix .8*), so the user can get a feedback while the device is worn. Kibana's web page runs on Javascript/Typescript and Hololens2's browser can't load the web page. We have found a workaround to this issue: a Python (Selenium [17]) script scrapes Kibana's homepage, downloads the graphs and upload them into a tiny web server based on Flask [18] Python library, accessible from Hololens2's browser (*See Appendix .9*).

Figure 3.5: Example of Kibana's dashboard.

Docker The back-end server system runs on a Docker multi container application: a *docker-compose.yml* configuration file will download and run every service and its dependency. This feature makes the application portable, since every computer equipped with docker can run the docker-compose.yml file.
See *Listing 3.4* for a *docker-compose.yml* file example.

```
1
2 version: "2.1"
3 services:
4     logstash:
5         build:
6             context: logstash
7             dockerfile: Dockerfile
8         container_name: logstash
9         volumes:
10            - ./logstash/csv:/usr/share/logstash/csv
11    kibana:
12        build:
13            context: kibana
14            dockerfile: Dockerfile
15        container_name: kibana
16        ports:
17            - 5601:5601
18        depends_on:
19            - elasticsearch
```

Listing 3.4: Docker-compose.yml example.

Chapter 4

System Evaluation

The proposed solution has been evaluated to verify whether the produced statistics actually reflect human behavior.

4.1 Proposed Solution vs Others Solutions

We first compare the Proposed Solution with **Holoframes**, a prototype of HololensObjectInteraction, and Hololens2 Research Mode [15] **StreamRecorderApp** [19].

Holoframes Holoframes is an early prototype of the Proposed Solution. Instead of using Hololens' API (PhotoCapture [24]), this version used to connect to Hololens2's web portal and take some screenshots. (*See Appendix .4*). This approach was replaced with a socket connection because of following issues:

- This approach did not consider hand joints and eye gaze coordinates.
- The screenshots were not synchronized due to network latency.

StreamRecorderApp Research mode enables access to raw streams on device. StreamRecorderApp is a sample app which shows how to capture many streams from Hololens2 (VLC cameras, Long throw depth, AHAT depth, PV, head tracking, hand tracking, eye gaze tracking), save them to the local disk and supply a set of python scripts for offline postprocessing. StreamRecorderApp is a powerful application, but it has some issues:

- It captures data streams and stores them to disk, but this is a batch process: the streams are not available for processing *until* the recording is stopped, not allowing to analyze data while the stream is opened.
- Since it tracks so many Hololens2's features, a few minutes of recording takes up to many GBs of storage, preventing to use the application for a long time, since it would fill up Hololens2's disk.

In *Table 4.1* there is a comparison of pros and cons of both approaches.

Software	✓Pros	✗Cons
HololensObjectInteraction	• Allows stream-analysis in a fast and efficient way • Does not store data on device, preventing Hololens2 to run out of storage. • tracks hand/eye gaze interactions	• Low stream frequency (0.5 Hz)
Holoframes	• Does not use socket connection, as a result the application on Hololens is more light • Does not store data on device, preventing Hololens2 to run out of storage.	• Does not track hands joint/eye gaze interactions • Network-dependent application • Low quality screenshots.
StreamRecorderApp	• Tracks so many stream sources • High stream frequency (30Hz)	• Does not track hands joint/eye gaze interactions • Only allows batch analysis, not real time analysis. • Storage consuming application.

Table 4.1: HololensObjectRecognition vs Other solutions.

4.2 Experience Survey

In order to evaluate the System, a survey has been conducted on 10 people. These 10 people had to complete some task in a artificial industrial laboratory, then they had to submit a survey on their experience. The object detector is trained to detect elements present in that artificial industrial laboratory. (*See Appendix .3*). *See Figures 4.1, 4.2.*

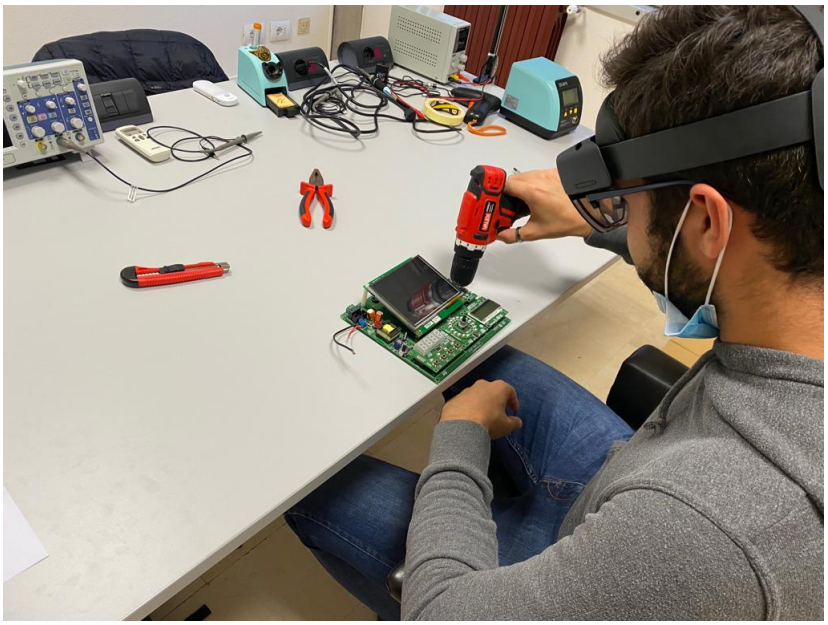


Figure 4.1: Scenario #4 test.

Figure 4.2: Scenario #5 test.

Testers had to stage the following 6 scenarios:

Scenario 1: Take the "scheda a basso voltaggio" board, turn the "saldatore" on, increase its temperature, act like you are welding, decrease its temperature, turn it off and put the board away.

Scenario 2: Take the "scheda ad alto voltaggio" board, turn the "alimentatore" on, connect cables to the board, disconnect cables to the board, turn the "alimentatore" down and put the board away.

Scenario 3: Take the "scheda a basso voltaggio" board, turn the "oscilloscopio" on, connect the "puntale oscilloscopio" to the board, disconnect the "puntale oscilloscopio" to the board, turn the "oscilloscopio" down and put the board away.

Scenario 4: Take the "avvitatore", make sure the "batteria avvitatore" is connected, take the "scheda a basso voltaggio" board, act like you are screwing something on the board, disconnect the "batteria avvitatore" to the "avvitatore" and put it away.

Scenario 5: Take the "pinza", put the "area lavoro" on the table, take the "scheda ad alto voltaggio" board, put it on the "area lavoro", act like you are pulling something away from the board, put the board and the "area lavoro" away.

Scenario 6: Put the "area lavoro" on the table, take the "registro", act like you are reading something from the "registro" book, put it and the "area lavoro" away.

In every stage there also were *distractor* objects, put there with the purpose of test whether the interactions would be recognized well.

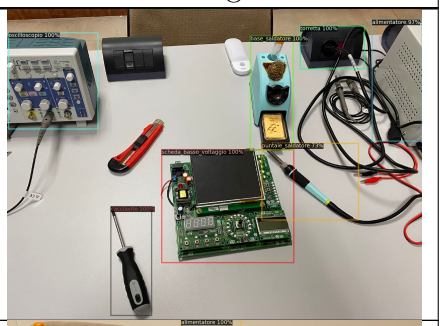
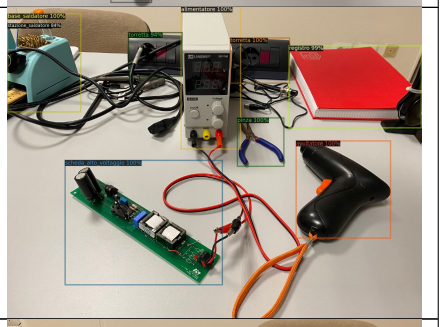
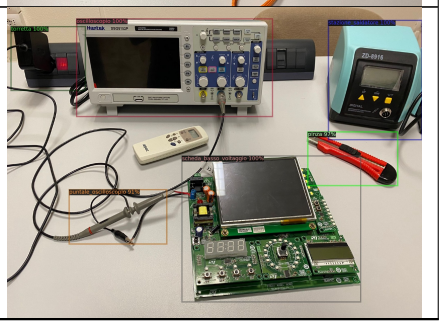
Scenario	Distractors	Image
Scenario 1	Cacciavite, Oscilloscopio	
Scenario 2	Registro, Avvitatore	
Scenario 3	Stazione saldatore	

Table 4.2: Scenarios.

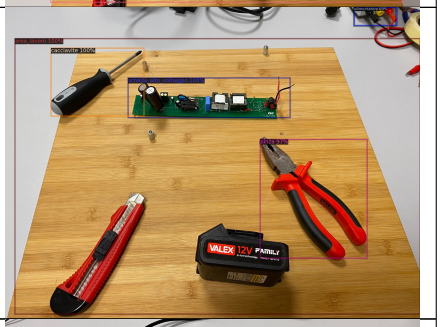
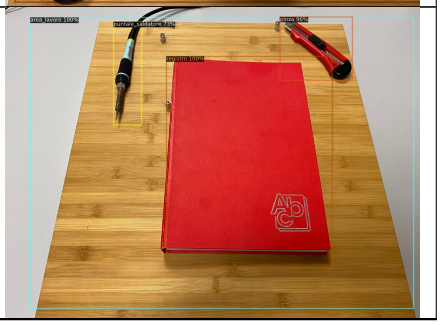
Scenario	Distractors	Image
Scenario 4	Cacciavite, Pinza	
Scenario 5	Cacciavite, Batteria avvitatore	
Scenario 6	Puntale saldatore	

Table 4.2: Scenarios.

See *Figure 4.3* for an example of scenario #3, *Figure 4.4* for an example of scenario #5 and *Figure 4.5* for an example of scenario #6. We can notice that, accordingly to the scenarios' setup:

- *Figure 4.3*: presents more "oscilloscopio" and "scheda_alto_voltaggio".
- *Figure 4.4*: presents more "scheda_alto_voltaggio" and "area_lavoro".
- *Figure 4.5*: presents more "registro", "torretta" and "area_lavoro".

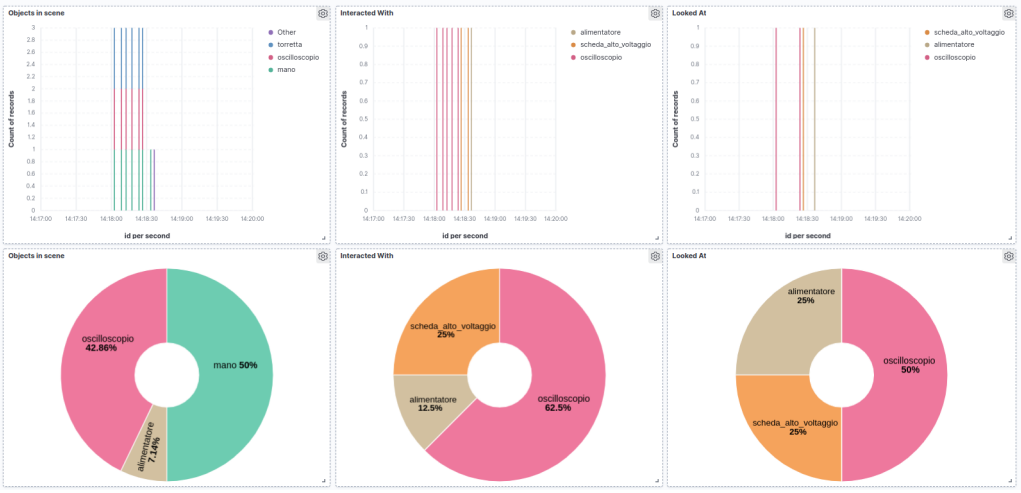


Figure 4.3: Scenario #3 Example.

Figure 4.4: Scenario #5 Example.

Figure 4.5: Scenario #6 Example.

4.3 Survey Results

The survey was structured in the following way:

1 Scenario x**:** Which of the following graphs is the most accurate?

2 Scenario x**:** Were the "Looked At" graphs accurate, according to your experience?

3 Scenario x**:** Were the "Interacted With" graphs accurate, according to your experience?

4 Scenario x**:** Were the "Object in scene" graphs accurate, according to your experience?

5 End of the survey: Did you find the dashboard insightful?

6 End of the survey: How do you rate User Experience?

See *Figure 4.6* for survey results to question 1. *Figure 4.7* for average score to questions 2, 3 and 4. See *Figure 4.8* for score to questions 5 and 6.

Figure 4.6 highlights that the most accurate graph, according to the testers experience, was "Interacted With". We expect this result due to the fact that hand tracking is usually much more accurate than gaze tracking, since hand tracking involves up to 10 points (10 finger tips) while eye gaze has only one point. Also note that Hololens2 has not been calibrated on the user's eye, hence gaze tracking may not work well.

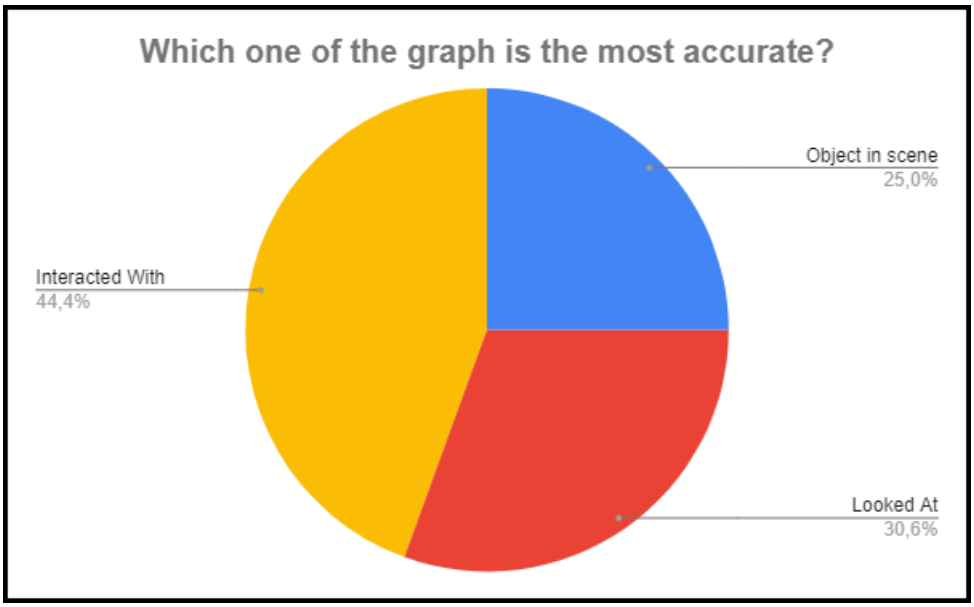


Figure 4.6: Which graph is the most accurate?

Figure 4.7 highlights that, still, the most accurate graph, according to the testers experience, was "Interacted With".
Its median and also first quartile are higher than other graphs .

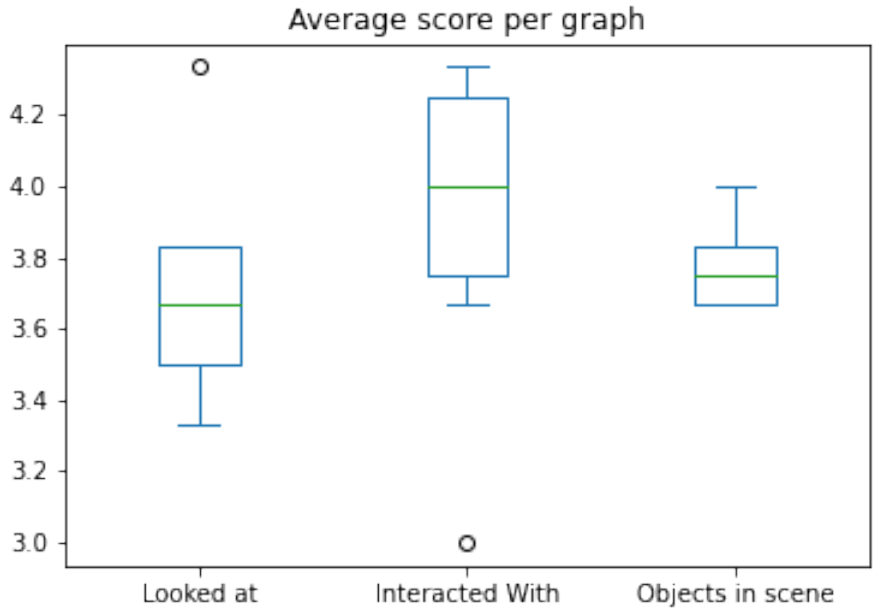


Figure 4.7: Were x graphs accurate, according to your experience?

Figure 4.8 shows that, according to the testers experience, the dashboard was very insightful and also the User Interface (buttons, web browser) was appreciated.

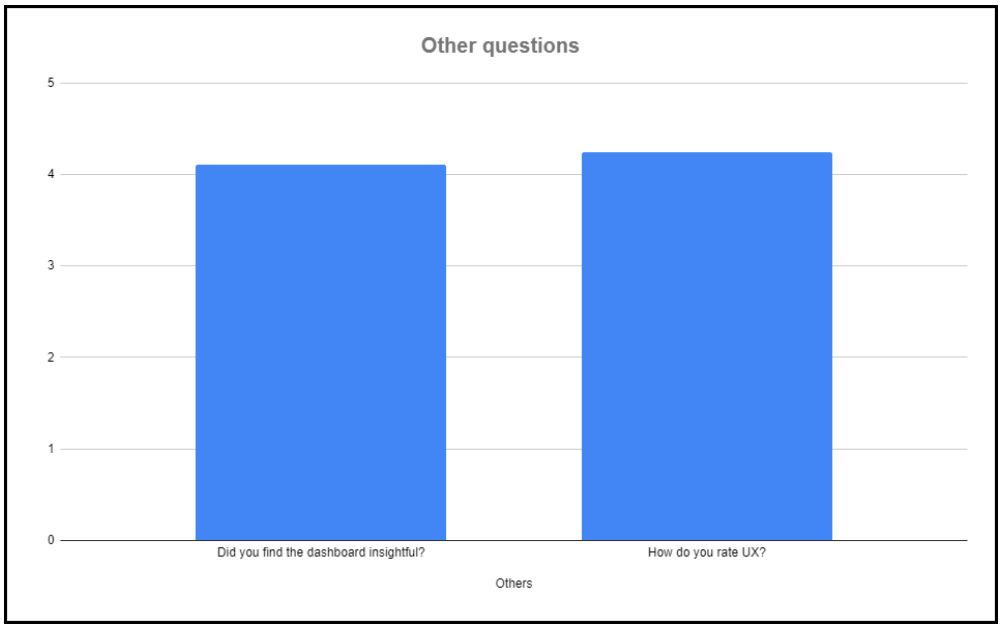


Figure 4.8: Other questions.

Conclusion

The proposed solution is a system which can be used to analyze real time data coming from Hololens2 device worn by a user in an industrial environment, in order to acknowledge his behaviour. The evaluation phase highlighted an overall good experience. The most accurate dashboard was the *Interacted With* one, because it is more accurate to track the 10 hand tips then the eye gaze.
Eye gaze tracking is not always accurate, which influences the looking at *Looked At* graphs results.
This system could be used inside a factory, to help increase productivity and also decrease occupational fatality.
However, the system is still a prototype, it has some issues and could be upgraded.
For instance:

- The 3D to 2D mapping of eye gaze and hand keypoints coordinates could be improved: at the moment this mapping uses static values, making the conversion not 100% accurate. It could be done with dynamic values coming from Hololens2's camera [20], which will make the interpolation much more accurate.

- Source code could be improved at a low level, in order to increase the frame rate, making the system more efficient.

- Some AR alerts could be added to Hololens2, in order to warn the user if he is not looking at the interacted object, which could be a dangerous behaviour in an industrial environment.

Appendices

.1 MRTK Setup

In order to program Hololens2 on Unity/Unity MRTK it necessary to setup the environment, download the required libraries and their dependencies. Here's the list of every software installed [23]:

- **Windows 10**.
- **Visual Studio 2019 (16.8 or higher)**.
  It is *mandatory* to install these extensions:

  ASP:NET and Web Development.

  .NET desktop development.

  C++ desktop development.

  UWP development.

  Unity game development.

  C++ game development.

  USB devices connectivity.

  Windows 10 SDK.
- **Windows 10 SDK**.
- **Unity 2019.4 LTS**.
  It is *mandatory* to install modules:

  Universal Windows Platform Build Support

  Windows Build Support (IL2CPP)
- **.NET 5.0 runtime (or higher)**.
- **Mixed Reality ToolKit**.

.2 Detectron2 on Google Colab

The first approach to Detectron2 [2] was on the Google Colab [21] platform. Google Colaboratory is a free Jupyter Notebook [22] which runs in the cloud, allowing to compute code on remote machines equipped with powerful GPUs and CPUs. Colab is designed for learning and testing, it is not implemented in this project because of the following issues:

- GPU usage is limited (*See Figure 9*).
- Time usage is limited (no more then 24 hours straight)
- It is not possible to automate tasks.

Figure 9: Colab ban for GPU-intensive use.

.3 Custom Object Detection Model

The custom detection model was trained on 3D models of real industrial tools. *See Table 3 for the full list of objects.*

Name	Type	Image
Alimentatore	Mobile	
Oscilloscopio	Mobile	
Stazione saldatore	Mobile	
Avvitatore	Mobile	

Table 3: Detectable Objects.

Name	Type	Image
Cacciavite	Mobile	
Pinza	Mobile	
Pinza	Mobile	
Puntale saldatore	Mobile	

Table 3: Detectable Objects.

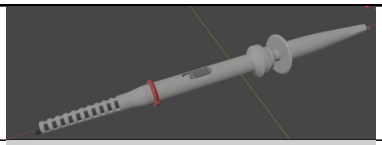
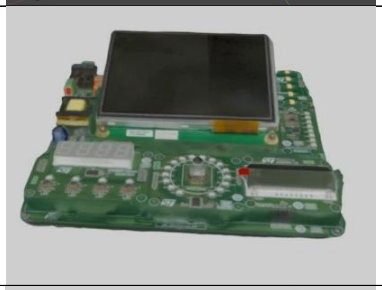
Name	Type	Image
Puntale oscilloscopio	Mobile	
Scheda basso voltaggio	Mobile	
Scheda alto voltaggio	Mobile	
Registro	Mobile	

Table 3: Detectable Objects.

Name	Type	Image
Batteria avvitatore	Mobile	
Area lavoro	Fixed	
Base saldatore	Mobile	

Table 3: Detectable Objects.

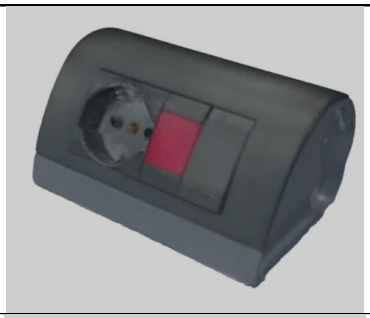
Name	Type	Image
Torretta	Fixed	
Pannello	Fixed	

Table 3: Detectable Objects.

.4 Holoframes

Holoframes (*See Listing 1*) is the prototype of HololensObjectInteraction. Instead of using Hololens' API (PhotoCapture [24]), this version used to connect to Hololens2's web portal and take some screenshots.

```
def connect(user, pw, addr):
    cap = cv.VideoCapture("https://"+ user +":"+ pw +"@"+addr+
    "/api/holographic/stream/live_high.mp4?holo=false&pv=true&
    mic=false&loopback=true")
    cap.set(cv.CV_CAP_PROP_FPS, 30) #limit fps to 30
    if not cap.isOpened():
        print("Cannot open camera")
        exit()
    return cap

def getFrames(cap, path):
    while True:
        ret, frame = cap.read()
        if not ret:
            print("Can't receive frame (stream end?). Exiting
    ...")
            break
        img = cv.cvtColor(frame, cv.COLOR_BGR2BGRA)
        timestamp = getTimestamp()
        cv.imwrite(os.path.join(path , timestamp+".png"), img)
        cv.imshow('frame', img)
        if cv.waitKey(1) == ord('q'):
            break
    cap.release()
    cv.destroyAllWindows()

if __name__ == "__main__":
    cap = connect()
    getFrames(cap)
```

Listing 1: Holoframes code.

.5 Socket on different Operating Systems

This project relies on a Socket connection located on heterogeneous systems, since the client is written in C# and runs on Hololens2, a Windows environment and the server is written in Python and runs on a remote Ubuntu environment.

C# Client: Hololens2's client sends a formatted json, containing hand joints/eye gaze logs and a photo (encoded in Base64 [16]).
See Listing 2.

Python Server: The remote server receives and decodes the json file, appends the logs onto a .csv file and saves the image.
Both the .csv file and the images are stored on a Docker Volume [9].
See Listing 3.

```
1  public class TCPTestClient
2  {
3    #region private members
4    private TcpClient socketConnection;
5    private Thread clientReceiveThread;
6    private NetworkStream stream;
7    #endregion
8    public static string IP;     //will be passed by scene 0
9    public static int PORT = 9999;
10
11   public void Start()
12   {
13     ConnectToTcpServer();
14   }
15
16   private void ConnectToTcpServer()
17   {
18     try
19     {
20       clientReceiveThread = new Thread(new ThreadStart(
             ListenForData));
21       clientReceiveThread.IsBackground = true;
22       clientReceiveThread.Start();
23     }
24     catch (Exception e)
25     {
26       Debug.Log("On client connect exception " + e);
27     }
28   }
29
30
```

```
31   private void ListenForData()
32   {
33     try
34     {
35       Debug.Log("Trying to connecto to :" + IP + ":" + PORT);
36       socketConnection = new TcpClient(IP, PORT);
37       stream = socketConnection.GetStream();
38     }
39     catch (SocketException socketException)
40     {
41       Debug.Log("Socket exception: " + socketException);
42     }
43   }
44   public void SendImage(byte[] image)
45   {
46     if (socketConnection == null)
47       return;
48     try
49     {
50       if (stream.CanWrite)
51       {
52         var imageBytesStr = Convert.ToBase64String(image);
53         string json = "{ \"msg\":" + "\"" + HandTracking.
               msgToSend + "\""+ ","
54           + "\"img\":" + "\"" + imageBytesStr + "\"" + "}";
55         byte[] jsonSize = Encoding.UTF8.GetBytes(json.Length.
               ToString().PadLeft(8, '0'));
56         stream.Write(jsonSize, 0, jsonSize.Length);
57         byte[] jsonStringToSend = Encoding.UTF8.GetBytes(json)
58         stream.Write(jsonStringToSend, 0, jsonStringToSend.
               Length);
59       }
60     }
61     catch (SocketException socketException)
62     {
63       Debug.Log("Socket exception: " + socketException);
64   }
65 }
```

Listing 2: Socket Client code.

```
HOST = '192.168.60.11' # Host address
PORT = 9999

def recv_json(conn, size):
    json_data = conn.recv(size, socket.MSG_WAITALL).decode("utf-8")
    parsed_data = json.loads(json_data)
    return parsed_data["msg"], base64.b64decode(parsed_data["img"])

def server_socket():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        print("listening on:" + HOST +":"+ str(PORT))
        s.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, 256000)
        s.bind((HOST, PORT))
        s.listen()
        conn, addr = s.accept()
        with open('csv/logs2D.csv', 'a', encoding='utf-8', newline='') as mynewfile:
            wr2D = csv.writer(mynewfile, quoting=csv.QUOTE_NONE, escapechar=' ')
            with conn:
                print('Connected by', addr)
                while True:
                    json_size = conn.recv(8).decode('utf-8')
                    print("json_size=" + json_size)
                    stringdata, img =recv_json(conn, int(json_size))
                    received_data = stringdata.split(' ')
                    row = received_data
                    new_row =[]
                    new_row = coord_converter(row[1:])
                    new_row.insert(0, row[0])
                    wr2D.writerow(new_row)
                    mynewfile.flush()
                    if not img: continue
                    try:
                        file_bytes = np.asarray(bytearray(io.BytesIO(img).read()), dtype=np.uint8)
                        imgToShow = cv2.imdecode(file_bytes, cv2.IMREAD_COLOR)
                        cv2.imwrite("csv/input_imgs/" + received_data[0] +'.jpg', imgToShow)
                    except Exception as e:
                        print("errore:", e)
```

Listing 3: Socket Server code.

.6 Hololens2's code improvement

Hololens2 has great potential, but has some hardware limitation, so it is fundamental to optimize code for the purpose of making the application lightweight. Here's the list of code improvements:

- Instead of using *Update()* functions, which are called 60 times a second, we call *InvokeRepeating()* because it allows to set the refresh rate, making the application more light.

- We set Unity quality settings [25] to *Very Low* on *Edit* menu > *Project Settings... > Quality.*
  This is not going to affect Holograms' quality, but it will decrease computation time.

.7 User Interaction

In order to detect an interaction, we suppose that whether hand tips/eye gaze coordinates are *near* an object's bounding box, that is named as a user interaction. So if there is an intersection between a hand tip coordinates +− an offset (10px) and the detected object bounding box, this is an interaction and the label of the detected object is returned. *See Listing 4.*

```
1     for i in range(len(outputs["instances"].pred_boxes)):
2         if((
3             (coord[0]-offset >= outputs["instances"].
    pred_boxes[i].tensor.cpu().numpy()[0][0] and
4              coord[0]-offset <= outputs["instances"].
    pred_boxes[i].tensor.cpu().numpy()[0][0] + outputs["
    instances"].pred_boxes[i].tensor.cpu().numpy()[0][2]) and
5            (coord[1]-offset >= outputs["instances"].pred_boxes[
    i].tensor.cpu().numpy()[0][1] and
6                 coord[1]-offset <= outputs["instances"].
    pred_boxes[i].tensor.cpu().numpy()[0][1] + outputs["
    instances"].pred_boxes[i].tensor.cpu().numpy()[0][3])
7         )
8             or
9             (coord[0]+offset >= outputs["instances"].
    pred_boxes[i].tensor.cpu().numpy()[0][0] and
10             coord[0]+offset <= outputs["instances"].
    pred_boxes[i].tensor.cpu().numpy()[0][0] + outputs["
    instances"].pred_boxes[i].tensor.cpu().numpy()[0][2]) and
11            (coord[1]+offset >= outputs["instances"].
    pred_boxes[i].tensor.cpu().numpy()[0][1] and
12             coord[1]+offset <= outputs["instances"].
    pred_boxes[i].tensor.cpu().numpy()[0][1] + outputs["
    instances"].pred_boxes[i].tensor.cpu().numpy()[0][3])
13        ):
14            if(pred_class_names[i] != "mano"):
15                lst.append(pred_class_names[i])
```

Listing 4: Check Interactions code.

.8 Async Web Browser

Since directly opening a web browser while the main application is running is going to cause its closure, when the user clicks on the browser button a web browser is launched as an asynchronus thread, not affecting the main application flow. (*See Listing 5*).

```
1  public void Open()
2          {
3  #if WINDOWS_UWP
4              UnityEngine.WSA.Application.InvokeOnUIThread(async
                   () =>
5              {
6                   System.Uri("http://www.google.com/"));
7                  bool result = await global::Windows.System.
                       Launcher.LaunchUriAsync(new System.Uri("
                       http://" + TCPTestClient.IP + ":5000"));
8                  if (!result)
9                  {
10                     Debug.LogError("Launching URI failed to
                           launch.");
11                 }
12             }, false);
13 #else
14             Application.OpenURL("http://" + TCPTestClient.IP +
                   ":5000");
15         }
```

Listing 5: OpenWebBrowser's code.

.9 Dashboards on Hololens2

In order to give a feedback to the user, a web browser has been implemented within the application. This web browser is meant to open Kibana's dashboard on Hololens2, but an issue occurred: Kibana's frontend is written in Javascript/Typescript and device's web browser could not render it. To overcome this problem, the solution provided was to use Selenium [17] for performing web scraping (*See Listing 6*) on Kibana's dashboard, save it as *.png* images and load them on a tiny web server based on Flask [18], which can be renderized from Hololens2(*See Listing 7*).

```
def scrape():
    driver = webdriver.Chrome(options=options)
    driver.set_window_size(1024, 600)
    driver.maximize_window()
    driver.get(kibana_dashboard_link)
    time.sleep(10)
    try:
        dismiss1 = WebDriverWait(driver, 1).until(EC.
    presence_of_element_located((By.XPATH,
    dismiss_button_location)))
        dismiss1.click()
        images = WebDriverWait(driver, 1).until(EC.
    presence_of_all_elements_located((By.XPATH,
    kibana_dashboard_div)))
        for i in range(0, len(images)(*-*)1):
            images[i].screenshot(imgs_directory)
    except:
        print("error")
    driver.close()

if __name__=='__main__':
    while True:
        scrape()
```

Listing 6: Selenium code.

```
app = Flask(__name__)
@app.route('/')
def get_images():
    result = [f for f in os.listdir("./static") if os.path.
    isfile(os.path.join("./static/", f))]
    html = '<meta http-equiv="refresh" content="11">'
    for x in result:
        html += '<img src="./static/{}"/>'.format(x)
    return html
```

Listing 7: Flask code.

Bibliography

[1] Microsoft Hololens2
https://www.microsoft.com/en-us/hololens

[2] Detectron2
Yuxin Wu and Alexander Kirillov and Francisco Massa and Wan-Yen Lo and Ross Girshick.
https://github.com/facebookresearch/detectron2

[3] Computer Vision
https://en.wikipedia.org/wiki/Computer_vision

[4] Unity
https://unity.com

[5] Mixed Reality ToolKit in Unity
https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity

[6] Industry 4.0
https://en.wikipedia.org/wiki/Fourth_Industrial_Revolution

[7] Internet Of Things
https://en.wikipedia.org/wiki/Internet_of_things

[8] Death by Working
https://www.weforum.org/agenda/2017/03/workplace-death-health-safety-ilo-fluor

[9] Docker
https://docs.docker.com/get-started/overview/

[10] Logstash
https://www.elastic.co/guide/en/logstash/current/index.html

[11] Apache Kafka
https://kafka.apache.org

[12] Apache Spark
https://spark.apache.org

[13] Elastic Search
https://www.elastic.co/elasticsearch

[14] Kibana
https://www.elastic.co/kibana

[15] Research Mode
https://github.com/microsoft/HoloLens2ForCV

[16] Base64
https://en.wikipedia.org/wiki/Base64

[17] Selenium
https://www.selenium.dev

[18] Flask
https://flask.palletsprojects.com

[19] StreamRecorderApp
https://github.com/microsoft/HoloLens2ForCV/tree
/main/Samples/StreamRecorder

[20] Computer Vision: Algorithms and Applications
Richard Szeliski
https://www.dropbox.com/s/8bf4feleifhrvl6
/SzeliskiBookDraft_20210930.pdf

[21] Google Colab
https://research.google.com/colaboratory

[22] Jupyter Notebook
https://jupyter.org

[23] MRTK Tutorial
Marco Ardizzone
https://github.com/marco-ardi/Mixed-Reality-ToolKit-Tutorial

[24] PhotoCapture
https://docs.unity3d.com/2019.4/Documentation/ScriptReference
/Windows.WebCam.PhotoCapture

[25] Unity recommended settings
https://docs.microsoft.com/en-us/windows/mixed-reality/develop/unity/recommended-settings-for-unity

[26] Faster R-CNN
https://arxiv.org/pdf/1506.01497.pdf

[27] Convolutional Neural Network
https://arxiv.org/pdf/1512.07108.pdf

[28] Region of Interest pooling
https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af