UNIVERSITÀ DEGLI STUDI DI CATANIA
Dipartimento di Matematica e Informatica
Corso di Laurea Triennale in Informatica

Davide Casano

HoloHelp: HoloLens Detection for a Guided Interaction

Relazione Progetto Finale

Relatore:
prof. Antonino Furnari

Anno Accademico 2020 - 2021

Abstract

The thesis has the primary aim to **help, assist and support people** to the usage of a specific object watched or interacted with. It does not claim to be used as a medical tool, but lays the groundwork for a different type of education in the future.
Starting from **Microsoft HoloLens 2** (a pair of mixed reality smartglasses), the idea was successfully implemented with a customized **Object Detector** trained in cloud via **Microsoft Azure**. To do that, we have used 9 specific classes of the **COCO dataset** to upload over 1000 images.
The goal is to make the tool as much manageable and easy to use as possible. Therefore, the idea was to use one simple voice command: *HoloHelp*. Nothing more. Once pronounced, HoloLens will immediately take a picture, saving the eye gaze coordinates of the object that the user is watching. Once saved the picture with those information, an API request will be sent to Microsoft Azure Custom Vision, and a small audio and AR video guide that explains the usage of the object detected will appear.
The application was tested by a sample of people who, with a set of questions, despite some limitations in terms of the number of classes available and efficiency of predictions, appreciated very much the tool and its usability.
The project has a great number of possible extensions that can be applied. Not considering technical improvements, the idea is to dedicate this kind of work to **people with impairments, labourers, children or elders**.

Contents

Chapter 1

Introduction

This thesis was designed starting with the primary idea of **helping people**. The project deals with assistance, support and aid user to the usage of objects. To develop the project, it has been used **HoloLens 2**: a mixed reality headset developed and manufactured by Microsoft.
The basic idea is to provide key information on the use of certain objects in front of the user, in order to help the user who may need this kind of assistance without calling for help from other people. That is the reason why the tool is called "**HoloHelp**" (*Figure 1.1: HoloHelp logo*).

Figure 1.1: HoloHelp Logo.

1.1 Goals

The aim of the project is to provide a **visual and audio description** of the elements that the user has in front of, in order to help and assist him to make the best choice.
The scenario that we consider is the one where a generic user has no familiarity or experience and wants to improve his knowledge on a specific object. HoloHelp assists in this particular context to **receive a guided usage** of the object with which the user has interacted (*Figure 1.4: a child using HoloHelp for washing his hands*).

1.2 Motivations

There is a large number of people for whom this tool can be useful and the topics that can be dealt with are many. First of all, **workplace safety** is nowadays fundamental. Only in Italy, in 2020, it has been registered more than 1200 deceased workers [1] at their place of work (*some statistics in Figure 1.2 and Figure 1.3*).
People performing a purely manual job are exposed to a considerable number of risks and security is never enough. In this regard, instruments and tools such as HoloHelp might have the possibility to support and help labourers to avoid running additional risks and increase their safety and productivity.

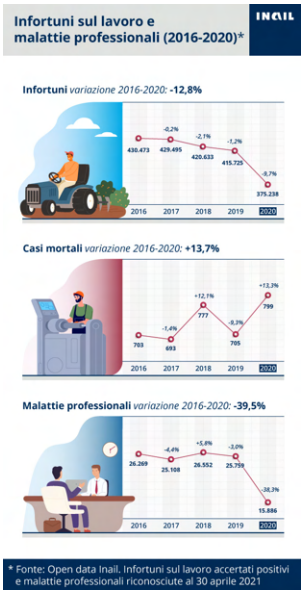

Figure 1.2: Work accidents and occupational diseases in Italy (2016–2020).

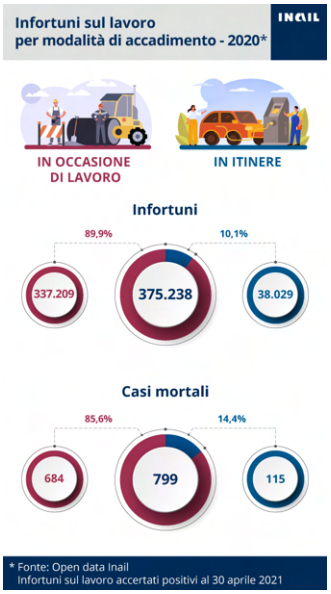

Figure 1.3: Work accidents by mode of occurrence (– 2020).

However, this tool is aimed not only at workers but also at **children**. Indeed, they cover a massive portion of persons who might need to receive information about the employment of a certain object. It is during the early school-age years that children begin to use the tools of their society with competence. In our culture, sometimes, electronic devices could be seen for children as something dangerous. This point of view is respectable for many reasons but, on the other hand, technology plays a key role in the educational field. Indeed, a child is often not capable in many sectors and, with the aid of a technological instrument, has the chance to improve his own skills and learn something new.

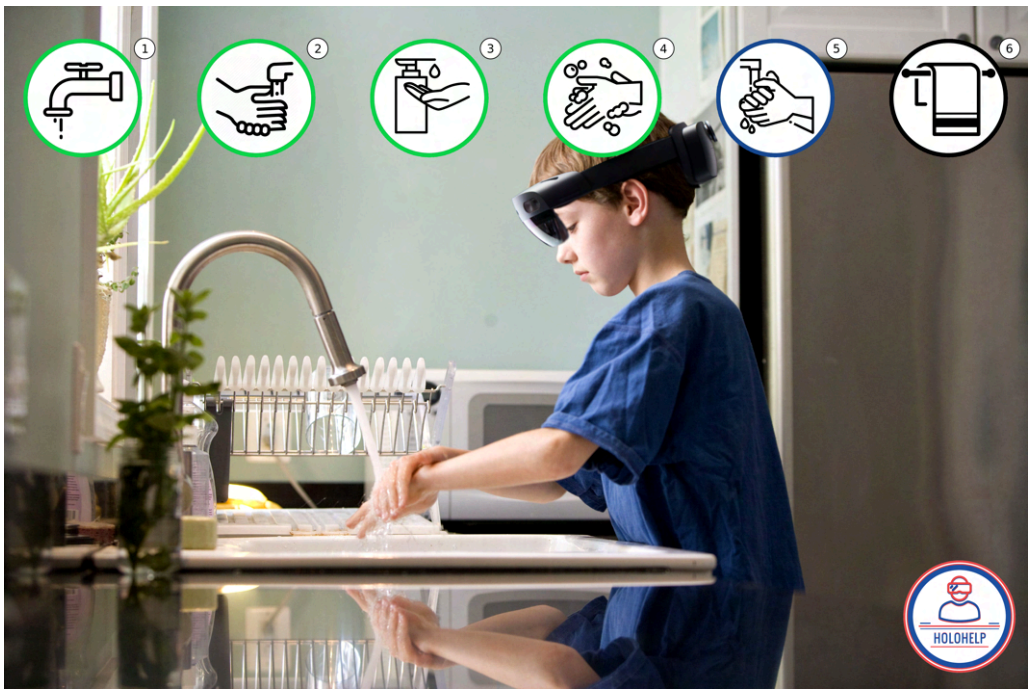

Figure 1.4: Child using HoloHelp.

Another portion of community to whom this tool could be useful is for **people with impairments**. This is a sensitive issue and HoloHelp doesn't pretend to be considered a medical device. However, the basic idea is to let this part of users be more confident and aware of the kind of object they communicate with.

1.3 Related Works

In this thesis it has been primarily used the Microsoft ecosystem. **Microsoft HoloLens 2**, for instance, is essential to the development of the work. **Microsoft Azure** has also played a fundamental role from the computational point of view.
Nevertheless, it is important to notice that Microsoft is not the only big tech company that is investing in this type of services. There is a remarkable kind of competitors that are expanding and investing in the Augmented Reality world.

1.3.1 A look into the future

Thanks to the speed of technology and engineering, we are going in a direction in which the **hardware covers less and less physical space** (*Figure 1.5: first 5MB Hard Drive made by IBM*). Indeed, it is not far a scenario where augmented reality devices like HoloLens will be within the reach of a large group of people not only in economic terms, but in terms of usability and ergonomics. In fact, it is very likely the realization of tools similar to smart glasses that can daily help the user in a normal lifestyle. Just think about the first hard drives.

Figure 1.5: First 5MB Hard Drive, 1956, IBM.

1.3.2 The Metaverse

The **metaverse** [2] is a hypothesized iteration of the internet, supporting persistent online 3D virtual environments through conventional personal computing, as well as virtual and augmented reality headsets (*Figure 1.6 shows Mark Zuckerberg fancing in the Metaverse*).
Current metaverse ambitions are centered on addressing technological limitations with modern augmented reality devices, as well as expanding the use of metaverse spaces to business, education, and retail applications.
Many entertainment and social media companies have invested in metaverse-related research and development. Within the education sector, it has been proposed that metaverse technologies would allow for more focused and interactive environments for learning history and human geography.

Figure 1.6: Mark Zuckerberg fencing in the Metaverse.

Despite all the pros that the introduction of the metaverse would bring (such as a more and immersive digital communication, a notable applications in businesses and a new electronic commerce, ...), the metaverse could obviously cause negative impacts in our society (for example addiction, losing track of time, psychological problems, ...). In short, this is a very recent field and, as everything related to the technological world, there should be a balanced and careful usage.

1.4 Thesis Structure

This thesis is structured as follows:

- Chapter 1: Thesis Introduction.
- Chapter 2: The chapter discusses the tools used in this thesis.
- Chapter 3: This chapter presents a full explanation of how the pipeline is organised.
- Chapter 4: This chapter reports an analysis on how the proposed solution is applied.
- Chapter 5: Thesis Conclusions.

Chapter 2

Methods and Tools

For the realization of this thesis a great number of tools has been used. **HoloLens 2** (*Figure 2.1: a picture of HoloLens 2*) has played various roles, among the main: source of data, acquisition of each frame and application's execution for displaying augmented reality content.

Figure 2.1: HoloLens 2.

Microsoft Azure Cognitive Services has been employed for data processing with the aid of the **COCO dataset** [3]. The latter, primarly employed for data image annotations, is connected to Microsoft Azure through **Python** APIs.
Unity is the graphics engine used for building 2D and 3D application, using **.NET** framework, **MRTK** (Mixed Reality ToolKit) packages and **C#** as programming language.
The following sections give details about the used tools.

2.1 Virtual Reality Headsets

A virtual reality headset is a head-mounted device that provides virtual reality for the wearer. **Virtual reality** (VR) **headsets** [4] are widely used with video games but they are also used in other applications, including simulators and trainers. They comprise a stereoscopic head-mounted display, stereo sound, and head-motion-tracking sensors, which may include devices such as gyroscopes, accelerometers, magnetometers or structured light systems.
VR glasses use a technology called head-tracking, which changes the field of vision as a person turns their head. The technology may not be perfect, as there is latency if the head moves too fast. Still, it does offer an immersive experience.

2.1.1 Microsoft HoloLens 2

Among all the tools, the one that plays one of the main roles is definitely **HoloLens 2**. This pair of glasses is used both as a source of source data (in the form of video and frames), and as a tool for visualizing output images (*Figure 2.2: a doctor using HoloLens 2 for a medical purpose*).

Figure 2.2: HoloLens 2 Usage.

HoloLens 2 is a mixed-reality electronic device that offers, in addition to a great number of high-tech hardware features (such as sensors or high resolution video cameras), security, scalability of cloud services and connection with artificial intelligence of **Microsoft Azure**.
The aim is to exploit this type of glasses as a data source for the acquisition of frames to be passed to the customized **object detection algorithm**. Once the computation is performed, it will be returned an image which, in short steps, explains how to use the identified object.

2.2 Object Detection

Object detection [5] in computer vision is the **ability to identify a specific object** in a sequence of images or videos. For each object, it is possible to extract a multitude of characteristics or information useful to provide a description of the object itself (*Figure 2.3: example of a generic object detection*).

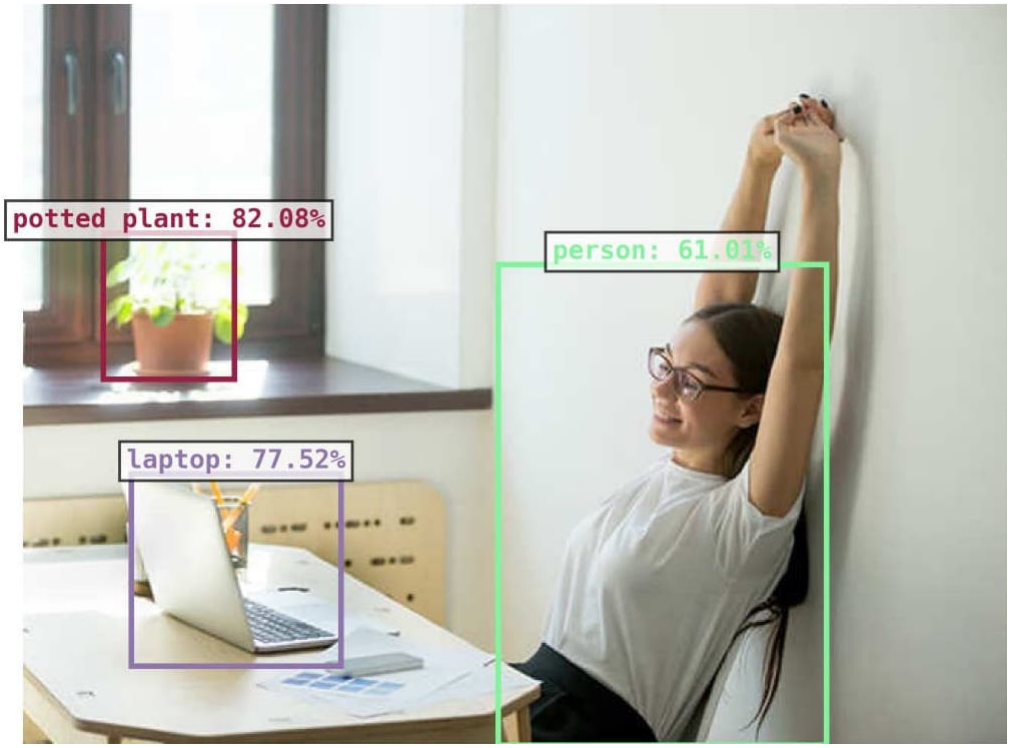

Figure 2.3: Object Detection Example.

Object detection is going to enter in a great number of fields, from personal safety to productivity at work. It is for this reason that object detection and recognition are applied in many areas of artificial vision, such as image recovery, security, healthcare and surveillance.

2.2.1 Main Applications

Some of the commonly asked questions that may arise when we are talking about object detection could be: what are its main applications? When should this approach be used? What are the advantages or benefits of using it?

- **Person Detection**: Person detection is the task of finding all the examples of individuals present in an image (*Figure 2.4: person detection made from a surveillance camera frame*). This model can be used for many purposes, some of these: counting the crowd (people counting), face detection and face recognition, smile detection or control of mask wearing (for COVID 19 security checks).

Figure 2.4: Person Detection Example.

It can be also commonly seen as the initial procedure in a video automated surveillance pipeline and deals with more significant level thinking modules as, for instance, actions recognition and dynamic scenes analysis.

- **Vehicle Detection**: The world moves faster and the number of cars continues to increase day by day. Vehicle recognition is very important in our daily life. It can be used for detecting the plate of a car in transit or of a car affected by an accident. The technique to carry out this type of operations is called **OCR** (**Optical Character Recognition**) (*Figure 2.5: OCR example of the '9' character*).

Figure 2.5: Optical Character Recognition.

On a road full of moving vehicles, this may therefore have an impact on both the safety of the company and the reduction in the number of traffic violations.

- **Tracking Objects**: an object detector framework can be also used for tracking object, for example tracking a ball during a football match, the swing of a cricket bat, a person in a video. Object tracking is also fundamental in video correspondence, robot vision and their activities (*see Figure 2.6 for a visual application of the object detection task in baseball*).

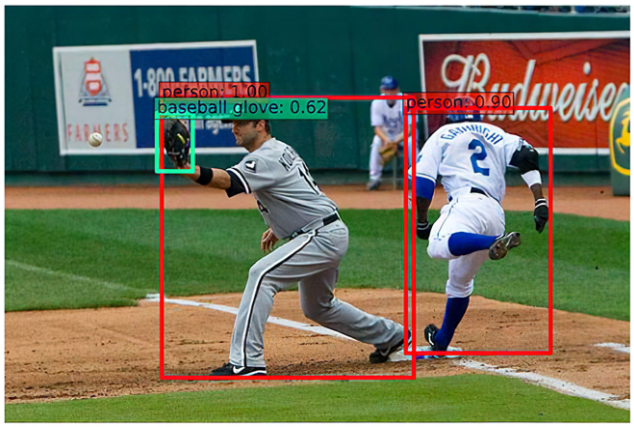

Figure 2.6: Object Detection in Baseball.

2.3 Unity

Unity [6] is a cross-platform graphics engine that allows the development of video games and other interactive content, such as architectural visualizations or 3D animations in real time. Unity development environment runs on Microsoft Windows, macOS and Linux (*Figure 2.7: Unity screenshot*).
The advantages of using this tool lay in the interoperability between scripts and in the **efficient connection with HoloLens 2**. Furthermore, through Unity we have realized a significant number of simulations, in order to avoid repeating and building each test on HoloLens 2.

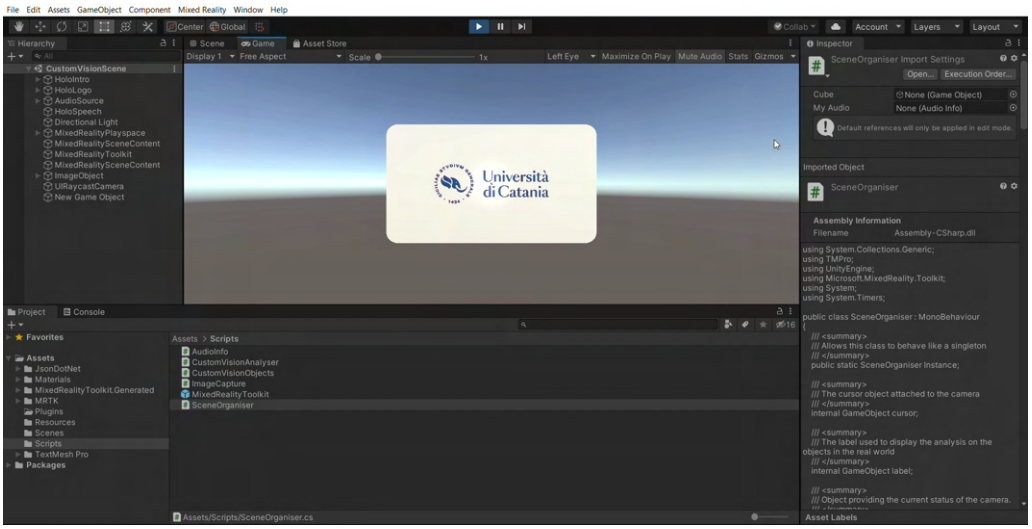

Figure 2.7: Unity HoloHelp screenshot.

2.4 .NET Framework

The **.NET Framework** [7] is a software framework developed by Microsoft that runs primarily on Microsoft Windows. It is used for programming in HoloLens, includes a large class library called Framework Class Library (FCL) and provides language interoperability (each language can use code written in other languages) across several programming languages (included C#, fundamental for Unity and HoloLens).

2.5 Mixed Reality Toolkit

Mixed Reality Toolkit (or **MRTK**) [8] is a collection of packages that enable cross platform Mixed Reality application development by providing support for Mixed Reality hardware and platforms. It is a Microsoft-driven project that provides a set of components and features, used to accelerate cross-platform MR app development in Unity. Here are some of its functions.

- Provides the cross-platform input system and building blocks for spatial interactions and UI.
- Enables rapid prototyping via in-editor simulation that allows you to see changes immediately.
- Operates as an extensible framework that provides developers the ability to swap out core components.
- Supports a wide range of devices: Microsoft HoloLens 1 and 2, Oculus Quest, Windows Mixed Reality headsets, iOS and Android devices.

2.6 Python and C#

Python and **C#** are the languages used for the entire project development. Through these, indeed, have been realized the upload of images from COCO dataset to Microsoft Azure Cognitive Services, the realization of the custom model of Object Detection and the script for uploading images to Hololens 2.

2.7 COCO Dataset

Datasets play a key role for the **training of the model** for the detection and classification of objects. **COCO** is a dataset published by Microsoft which provides detection, segmentation and annotations in wide scale.
The chosen dataset for this thesis was published in 2017 and provides **90 common object classes**, organised into different supercategories: person, vehicle, animal, accessory, sports, food, appliance, ... (*see Listing 3 for the entire list of "categories"*).
For the creation of the customized trained model it has been used a .JSON file, which contains different kinds of annotations.

2.8 Microsoft Azure Cognitive Services

Microsoft Azure Cognitive Services [9] are cloud-based services with REST APIs and client library SDKs available to help you build cognitive intelligence into your applications. You can add cognitive features to your applications without having artificial intelligence (AI) or particular data science skills. Azure Cognitive Services comprise various AI services that enable you to build cognitive solutions that can see, hear, speak, understand, and even make decisions. It puts artificial intelligence within the reach of any developer. Everything is managed through API calls in Python, useful to work at a higher level and recall methods for the development of your object detection model (*Figure 2.8: a generic Microsoft Azure pipeline*).

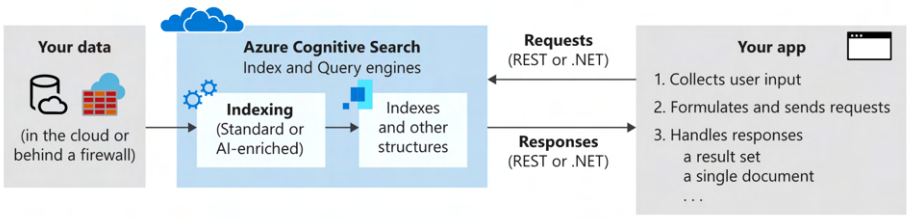

Figure 2.8: Microsoft Azure pipeline.

2.8.1 Custom Vision

Custom Vision [10] is an artificial intelligence service offered by Microsoft Azure that allows you to create, deploy and improve image prediction models. It works using a customized *object detector*, which applies labels to images according to the detected features. Unlike classic computer vision services,

indeed, **Custom Vision allows to specify your own labels and train custom models** to detect them. For this purpose, the interaction between COCO and Azure is therefore fundamental (*Figure 2.9: a generic Microsoft Azure Custom Vision pipeline*).

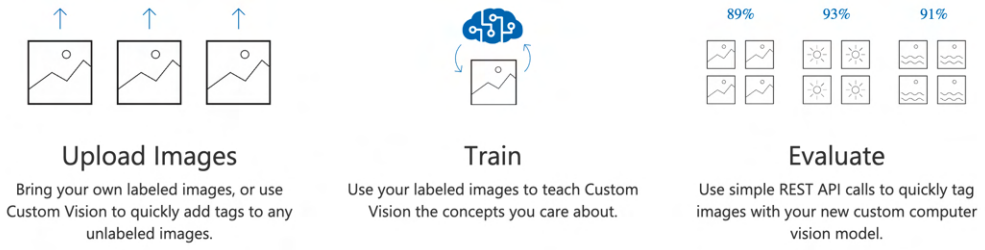

Figure 2.9: Microsoft Azure Custom Vision pipeline.

Chapter 3

Proposed Solution

The basic concept is to use HoloLens 2 as an instrument to detect objects in real time in order to **inform and support the user** who is employing this kind of tool. To do that, the proposed solution considers different approaches (*Figure 3.1 illustrates the generic Machine Learning pipeline that has been employed for the development of the project*).
In this scenario, the pipeline is structured as follows:

1. **Data Acquisition**: images preparation, uploading and extraction.
2. **Model Building**: training and test of the model.
3. **Application Prediction**: prediction and guide of each photo sent to the model.

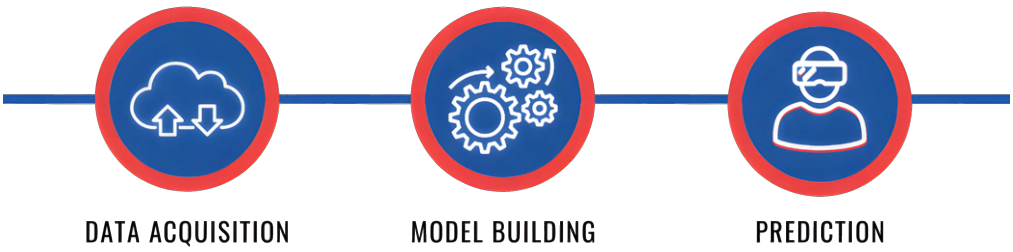

Figure 3.1: Machine Learning Pipeline followed.

The full code is available from GitHub: github.com/davide-cas/HoloHelp.

3.1 Data Acquisition

Firstly, it has been downloaded the **2017 COCO .JSON file** which contains a lot of key information: *info*, *licenses*, *images*, *annotations* and *categories*. All this set of data is relevant, but for the upload of all the images on the Microsoft Azure Cognitive Services portal were treated only:

- *images*.
- *annotations*.
- *categories*.

3.1.1 Images

A generic JSON "*images*" *i-th* value contains:

```
{
  'license': 1,
  'file_name': '000000153299.jpg',
  'coco_url': 'http://images.cocodataset.org/val2017/000000153299.jpg',
  'height': 500,
  'width': 461,
  'date_captured': '2013-11-17 01:35:43',
  'flickr_url': 'http://farm1.staticflickr.com/88/233835328_6c4490cf34_z.jpg',
  'id': 153299
}
```

Listing 1: COCO .JSON "images" example.

Here, the only information taken into account is the "*coco_url*". The idea is to avoid the download and storing in memory of each photo. Everything is related to the cloud, **organized and managed via Microsoft Azure Custom Vision**. The image is so sent via URL with a simple script (*see Listing 3.2 to see a part of the full script used to upload the COCO's photos to the Microsoft Azure model*).

3.1.2 Annotations

A generic JSON "*annotations*" *i-th* value contains:

```
{
    'segmentation': [[
       510.66,
       423.01,
       511.72,
       420.03,
       510.45,
       416.0,
       510.34,
       413.02,
       510.77,
       410.26,
       510.77,
       407.5,
       ...]],
     'area': 702.105,
     'iscrowd': 0,
     'image_id': 289343,
     'bbox': [473.07, 395.93, 38.65, 28.67],
     'category_id': 18,
     'id': 1768
}
```

Listing 2: COCO .JSON "annotations" example.

Here, the key entries taken into consideration are:

- *image_id*: represent the id of the image.
- *bbox*: a bounding box is the smallest rectangle with vertical and horizontal sides that completely surrounds an object. Starting from the left side of the image, the **first value** is equal to the distance between the top-left corner of the object and the left side itself. The same reasoning applies to the **second value**, but starting from the top. The **third** and **fourth value** are, instead, respectively the width and height of the created rectangle.
- *category_id*: corresponds to the specific id of the category of the single annotation (each photo can contain multiple annotations).

3.1.3 Categories

The JSON "*categories*" key value contains:

```
{
     {'supercategory': 'person', 'id': 1, 'name': 'person'},
     {'supercategory': 'vehicle', 'id': 2, 'name': 'bicycle'},
     {'supercategory': 'vehicle', 'id': 3, 'name': 'car'},
     ...
     {'supercategory': 'outdoor', 'id': 10, 'name': 'traffic light'},
     {'supercategory': 'outdoor', 'id': 11, 'name': 'fire hydrant'},
     ...
     {'supercategory': 'animal', 'id': 16, 'name': 'bird'},
     {'supercategory': 'animal', 'id': 17, 'name': 'cat'},
     ...
     {'supercategory': 'accessory', 'id': 27, 'name': 'backpack'},
     {'supercategory': 'accessory', 'id': 28, 'name': 'umbrella'},
     ...
     {'supercategory': 'sports', 'id': 34, 'name': 'frisbee'},
     {'supercategory': 'sports', 'id': 35, 'name': 'skis'},
     ...
     {'supercategory': 'kitchen', 'id': 48, 'name': 'fork'},
     {'supercategory': 'kitchen', 'id': 49, 'name': 'knife'},
     ..
     {'supercategory': 'food', 'id': 52, 'name': 'banana'},
     {'supercategory': 'food', 'id': 53, 'name': 'apple'},
     ...
     {'supercategory': 'furniture', 'id': 62, 'name': 'chair'},
     {'supercategory': 'furniture', 'id': 63, 'name': 'couch'},
     ...
     {'supercategory': 'electronic', 'id': 72, 'name': 'display'},
     {'supercategory': 'electronic', 'id': 73, 'name': 'laptop'},
     {'supercategory': 'electronic', 'id': 74, 'name': 'mouse'},
     {'supercategory': 'electronic', 'id': 75, 'name': 'remote'},
     {'supercategory': 'electronic', 'id': 76, 'name': 'keyboard'},
     {'supercategory': 'electronic', 'id': 77, 'name': 'cell phone'},
     {'supercategory': 'appliance', 'id': 78, 'name': 'microwave'},
     {'supercategory': 'appliance', 'id': 79, 'name': 'oven'},
     {'supercategory': 'appliance', 'id': 80, 'name': 'toaster'},
     {'supercategory': 'appliance', 'id': 81, 'name': 'sink'},
     {'supercategory': 'appliance', 'id': 82, 'name': 'refrigerator'},
     {'supercategory': 'indoor', 'id': 84, 'name': 'book'},
     {'supercategory': 'indoor', 'id': 85, 'name': 'clock'},
     ...
}
```

Listing 3: COCO .JSON "annotations" example.

In this model there are 9 classes used for the upload, with a total of **1364 uploaded images** (*see Table 4.1 for the entire list of the classes chosen*).

	Class	no. Uploaded Images
1	cell phone	214
2	display	207
3	sink	186
4	laptop	183
5	knife	181
6	remote	145
7	keyboard	106
8	mouse	88
9	microwave	54

Table 3.1: Entire list of classes selected.

NB: the model has been trained using 1364 images, sufficient amount of images to get an excellent level of performance. This limit is due to the fact that an academic license has been used for the project. In this regard, therefore, it is important to keep in mind that some predictions might have low levels of recall and therefore not be exactly true to the desired output. The pie chart shown in *Figure 3.2* illustrates the values of **precision**, **recall** and **mAP**.

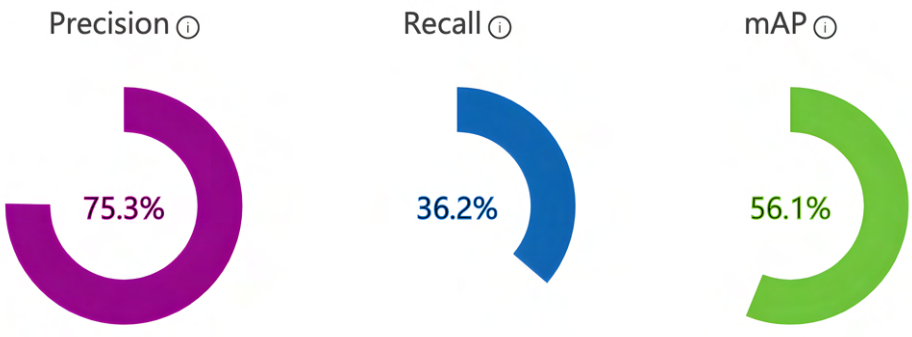

Figure 3.2: Precision, Recall and mAP values.

3.2 Model building

Once selected the entire list of classes that the model provides, the following step is to **physically upload the photos to the Microsoft Azure Custom Vision's portal** using an API connection.

3.2.1 API Connection

At first, three information were saved from the official **Microsoft Azure Custom Vision site**: *training key*, *prediction key* and *endpoint*. These information are accessible from the *Project Settings* heading (*see Figure 3.3*).

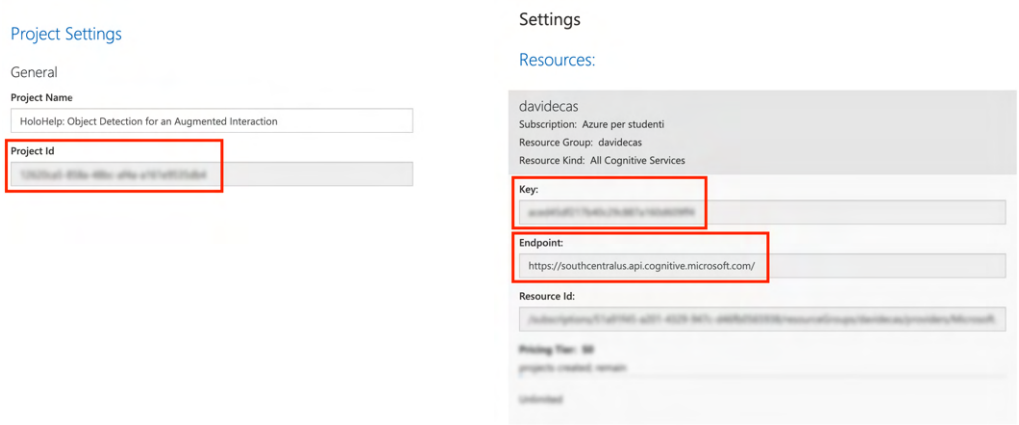

Figure 3.3: Custom Vision API Keys.

The upload of 1364 images has been realized **through a Python script**. The following code snippet (*Listing 3.2*) shows the API Connection and the specific libraries used to make it.

```
1 # Importing Azure's libraries
2 from azure.cognitiveservices.vision.customvision.training
    import CustomVisionTrainingClient
3 from azure.cognitiveservices.vision.customvision.prediction
    import CustomVisionPredictionClient
4 from azure.cognitiveservices.vision.customvision.training.
    models import ImageFileCreateBatch, ImageFileCreateEntry,
    Region
5 from msrest.authentication import ApiKeyCredentials
```

```
# Making the connection using: Training Key, Prediction Key and Endpoint.
credentials = ApiKeyCredentials(in_headers={``Training-key'': training_key})
trainer = CustomVisionTrainingClient(ENDPOINT, credentials)
prediction_credentials = ApiKeyCredentials(in_headers={``Prediction-key'': prediction_key})
predictor = CustomVisionPredictionClient(ENDPOINT, prediction_credentials)
```

Listing 3.1: Microsoft Azure Custom Vision API connection in Pyhon.

3.2.2 Photos Upload

Once accessed to the Custom Vision APIs, it is possible to upload the images contained in the .JSON file. Through the following Python script snippet (*Listing 3.2*), we have saved the selected categories, url, annotations and bounding box of different images.

```
sel_categories = [
    "remote",
    "mouse",
    "laptop",
    "keyboard",
    "tv",
    "microwave",
    "sink",
    "toaster",
    "knife",
    "cell phone" ]
# Reading each category
for i in range(len(sel_categories)):
    for j in range(n_categories):
        if sel_categories[i] == data["categories"][j]["name"]:
            category_ids.append(data["categories"][j]["id"])
            break
# Saving annotations, bbox and image id
for image_index in image_indexes:
        urls.append(data["images"][image_index]["coco_url"])
        image_id = data["images"][image_index]["id"]
        annotations = [y["bbox"] for y in data["annotations"] if (y["image_id"] == image_id) and (y["category_id"] == category_ids[x])]
        bbox.append(annotations)
        imgs.append(data["images"][image_index])
```

Listing 3.2: Python code snippet used for the selection of the photos.

3.2.3 Training of the Model

From this moment, it is possible make the training of the model. Microsoft Azure Custom Vision provides different approaches to train the images uploaded. The one that **HoloHelp** uses **is based on a 3 hours training** (*see Figure 3.4 for the Microsoft Custom Vision training options*). This is due to the fact that a low number of images were used and, consequently, the training lasts less.

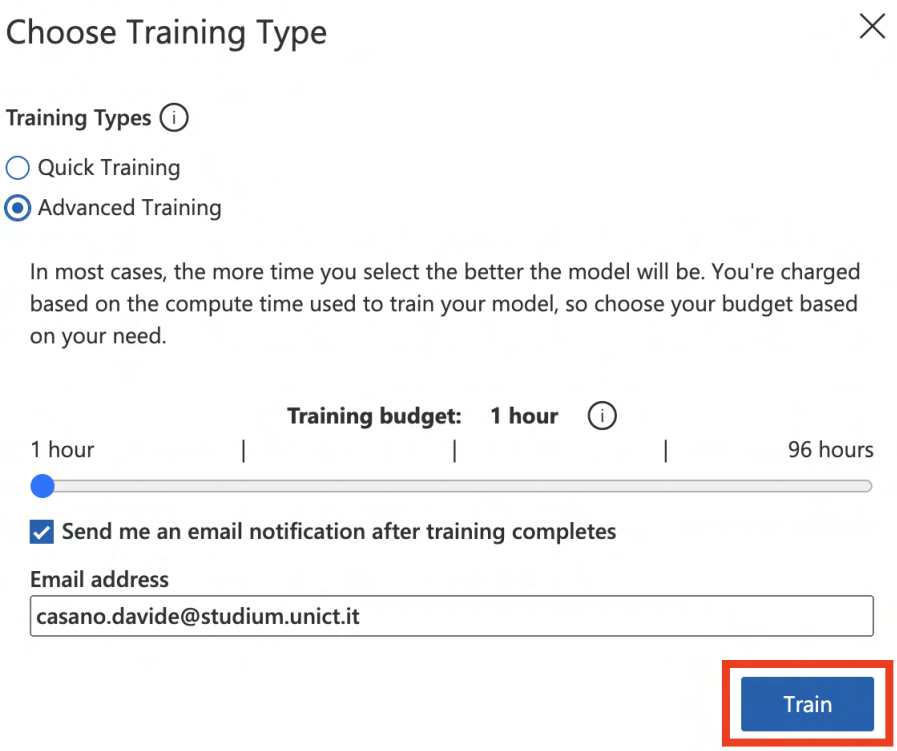

Figure 3.4: Microsoft Custom Vision training options.

3.2.4 Test of the Model

Starting from this point, **the model is ready to be used**. It is possible send an image to get the detection results (*see Figure 3.5 to see a cell phone detection*).

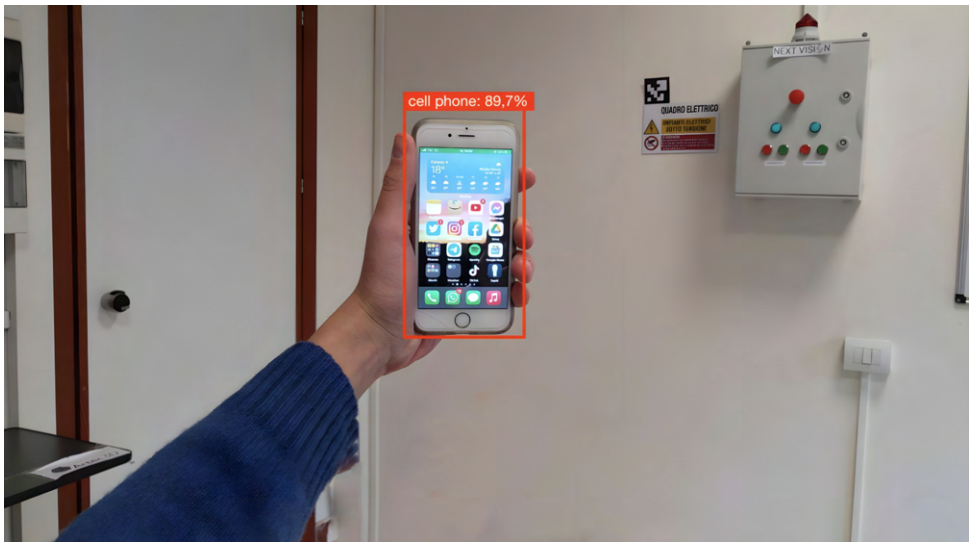

Figure 3.5: Cell Phone detection.

3.3 Application Prediction

In HoloLens everything is arranged with a voice command, **a specific keyword**: "***HoloHelp***". Once pronounced this word, HoloLens will automatically make a photo. The image is locally saved and sent through the API connection. The following C# script (*Listing 3.3*) shows the process of image capturing and sending to Azure Custom Vision.

```
1  private void ExecuteImageCaptureAndAnalysis()
2  {
3      // Update camera status to analysis.
4      SceneOrganiser.Instance.SetCameraStatus("Analysis");
5
6      // Create a label in world space using the SceneOrganiser
           class
7      SceneOrganiser.Instance.PlaceAnalysisLabel();
8
9      // Set the camera resolution
10     Resolution cameraResolution = UnityEngine.Windows.WebCam.
           VideoCapture.SupportedResolutions.OrderByDescending((
           res) => res.width * res.height).First();
11
12     Texture2D targetTexture = new Texture2D(cameraResolution.
           width, cameraResolution.height);
13
```

```
14      // Begin capture process, set the image format
15      UnityEngine.Windows.WebCam.PhotoCapture.CreateAsync(true,
            delegate (UnityEngine.Windows.WebCam.PhotoCapture
            captureObject)
16      {
17          photoCaptureObject = captureObject;
18          UnityEngine.Windows.WebCam.CameraParameters
                camParameters = new UnityEngine.Windows.WebCam.
                CameraParameters
19          {
20              hologramOpacity = 0.5f,
21              cameraResolutionWidth = targetTexture.width,
22              cameraResolutionHeight = targetTexture.height,
23              pixelFormat = UnityEngine.Windows.WebCam.
                    CapturePixelFormat.BGRA32
24          };
25
26          // Capture the image from the camera and save it in
                the App internal folder
27          captureObject.StartPhotoModeAsync(camParameters,
                delegate (UnityEngine.Windows.WebCam.PhotoCapture.
                PhotoCaptureResult result)
28          {
29              string filename = string.Format(@"CapturedImage
                    {0}.jpg", captureCount);
30              filePath = Path.Combine(Application.
                    persistentDataPath, filename);
31              captureCount++;
32              photoCaptureObject.TakePhotoAsync(filePath,
                    UnityEngine.Windows.WebCam.
                    PhotoCaptureFileOutputFormat.JPG,
                    OnCapturedPhotoToDisk);
33          });
34      });
35  }
```

Listing 3.3: ExecuteImageCaptureAndAnalysis C# Script.

3.3.1 Eye Tracking

In addition to the photo taken, **the tool saves information about the eye tracking**. In particular, from the C# code, it is called 60 times per second the *Update* function, which refreshes and update the new gaze position. Once obtained the 3D coordinates, it is made a mapping from the 3D to the 2D world (*see Appendix .1 for the full explanation of the script*).
Getting the gaze coordinates is crucial for the usability of the tool.

Indeed, the idea is to avoid all the commands related to actions, in order to improve the user experience and do not make the user feel disoriented with the headset.

3.3.2 Communication Approaches

The primary idea is to assist and help users. To do that, **three different kinds of communication approaches** have been implemented:

- **feedback audio**: through a female voice, the *Text To Speech* (TTS) software feature it has been exploited. TTS is capable of converting almost any text-based message into an easily understood verbal message.
- **feedback video**: it simply uses a video to explain how use a specific object watched.
- **feedback via photo**: similar to the video feedback, it consists of a set of linked images. The idea is to show each photo for 3/4 seconds on average.

3.3.3 Application Components

Here is a simple flow chart that shows the different components of the application (*see Figure 3.6*).

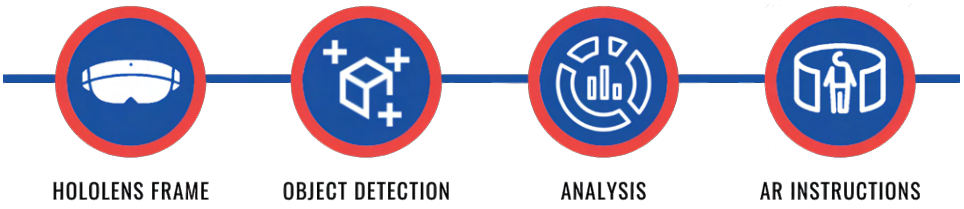

Figure 3.6: Application Components' Pipeline.

The first step is to **take a photo using HoloLens**. Secondly, the photo is sent through an API connection to Microsoft Azure in order to perform **object detection**. The third step is to do the **analysis** so as to understand what the object being observed is. The last step is to show a **brief guide** on HoloLens of the object looked at or interacted with.

3.4 Examples

Once the object has been detected, information about the recognized object will be displayed. To see how the tool works and what kind of explanations have been used, here are some examples of frames taken in first person from HoloLens 2.

3.4.1 First Example: Sink

When HoloHelp is pronounced and the object watched is a sink, the tool would quickly send the photo to Custom Vision. If the prediction is correct, HoloHelp will explain how to wash your hands step-by-step (*see Figure 3.7*).

Figure 3.7: Sink detection.

3.4.2 Second Example: Cell Phone

If the recognized object is a mobile phone, it will be displayed a video of a person who is unlocking his phone (*see Figure 3.8*).

Figure 3.8: Cell Phone detection.

3.4.3 Third Example: Display

If the recognized object is a display, it will be visualized a tutorial of how step-by-step turning on a generic display (*see Figure 3.9*).

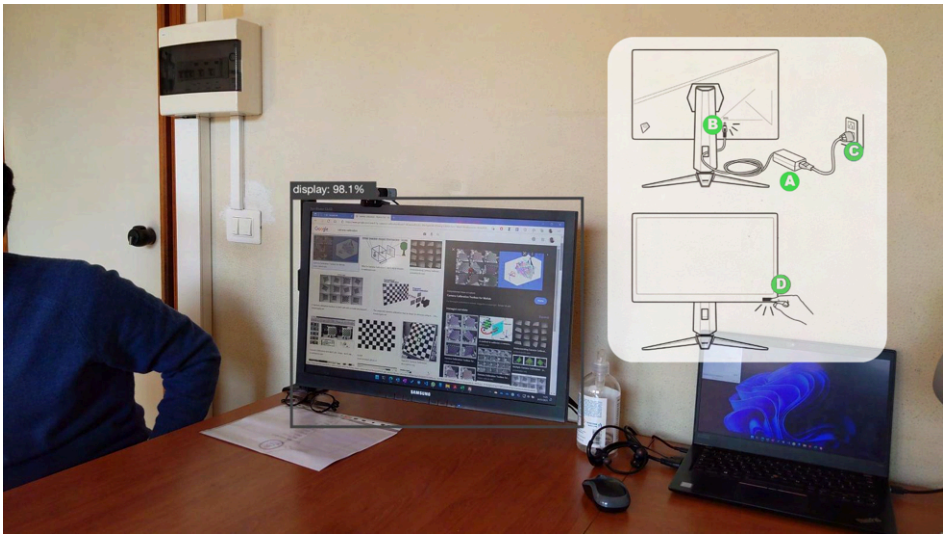

Figure 3.9: Display detection.

3.4.4 Fourth Example: Mouse

If the recognized object is a mouse, it will be visualized a generic guide for what a mouse is useful and where is used (*see Figure 3.10*).

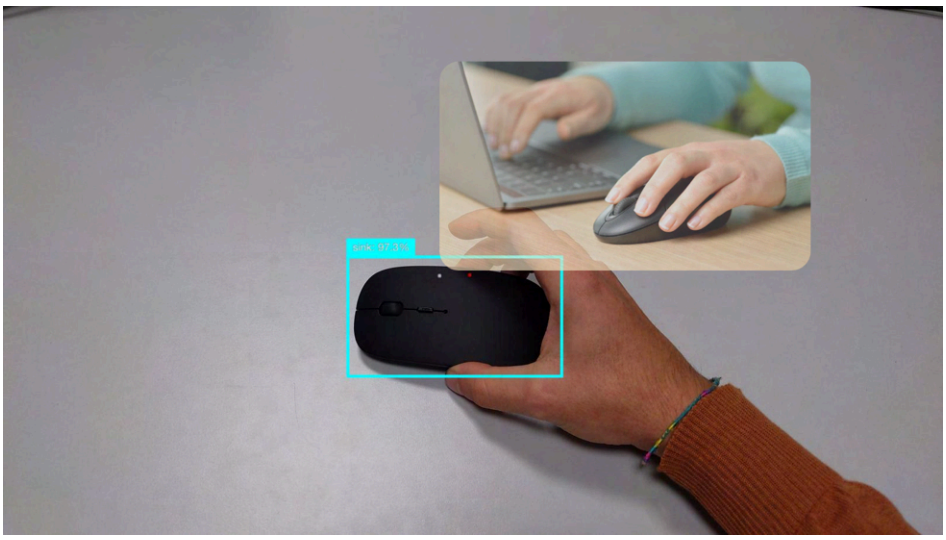

Figure 3.10: Mouse detection.

Chapter 4

System Evaluation

4.1 Different Approaches

For the realization of HoloHelp tool two different approaches have been considered. The first, uses **the closest object to the center**; the second is actually an upgrade of the first one and **exploits the eye tracking**.

4.1.1 Closest Object to the Center

This is the first solution used, **a sort of "alpha" version of HoloHelp**. The reason why this decision has been taken is due to the fact that, at first, the gaze was not employed. It consists in comparing all the bounding boxes present in the image, in order to **find the closest to the center** with the assumption that this would be the object watched.
This approach is actually wrong for many reasons. In general, it could cause a great number of misunderstandings, even though it has a certain stability on the usage (unless object detector errors, always returns the class closest to the center). *Figure 4.1* shows a "+" symbol and a *red and yellow circle*. The first indicates the center of the image, the second shows where the user was watching at the moment of the photo was taken. In a typical scenario of different objects that a user might have in front of, even if the object observed is actually the *cell phone*, there are returned information about *keyboard* because its bounding box is the closest to the center.

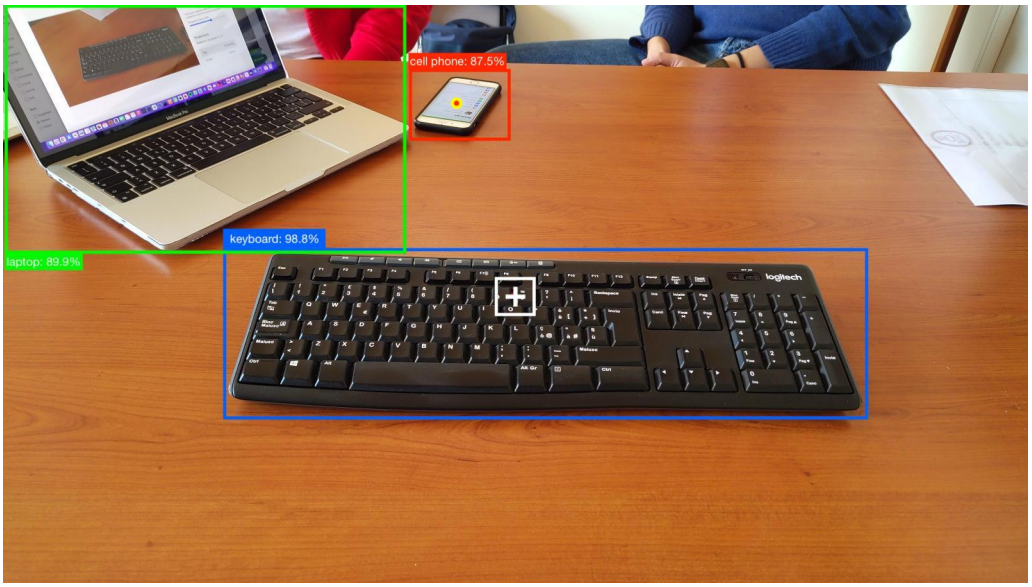

Figure 4.1: Alpha version of HoloHelp, returning the closest object to the center.

4.1.2 Eye Tracking

This solution arises from a principle of psychology for which, before interacting with an object this is typically watched by the user. For this reason, we considered the eye tracking. This approach is really useful if we think about the usability of the tool. Indeed, now the tool no longer depends on the position of the objects in scene but only on the gaze. The object for which information will be returned after taking the photo **is only the one watched** (for instance, in a similar *Figure 4.1* scenario, the object returned will be the cell phone).

4.1.3 Comparison between the two approaches

Both solutions have pros and cons. The alpha solution always returns the closest object to the center. As it was said before, this approach provides **stability** and the usage of an **unique metric**. At the same time, nevertheless, the **information** and **objects returned** in all probability might be wrong.
If we want to be precise, the eye tracking approach is certainly better than the first version, because provides a notable efficiency and precision in the output of the recognized objects. On the other hand, it calls the *updateGaze* (*see Appendix .1*) function 60 times per seconds (in order to update the new position of the Gaze), so is in a certain way **expensive**. Furthermore, the

mapping is not 100% accurate, reason why is needed a calibration before the usage (*see Appendix .2 for more information*) and it has been introduced an offset to each recognized object (*see Appendix .3 for more details*). This means that the solution could rarely return the wrong object.

Alpha Solution		**Eye Tracking Solution**	
Pros	**Cons**	**Pros**	**Cons**
stability	object returned	efficiency	expensive
unique metric	information returned	object returned	mapping 3D to 2D

Table 4.1: Pro and Cons summarised.

4.2 Survey

The primary function of HoloHelp is to assist people. To make the best for the users it is decisive asking somebody who have tested the tool if he/her have appreciated it and what could be changed in a future version. To do that, **it was created a simple survey filled in by 9 people**. Questions and answers are structured as follows:

1. Considering the users to whom the tool is aimed, how useful is the tool?
2. How accurate do you think object detection is?
3. Among the detected objects, which explanation do you believe to be the most accurate?
4. How clear were the explanations?
5. How useful did you find the audio feedback?
6. How useful did you find the video feedback?
7. How useful did you find the photo usage for the explanation?

4.2.1 Survey Outcome

1. Considering the users to whom the tool is aimed, how useful is the tool? (*Figure 4.2*)

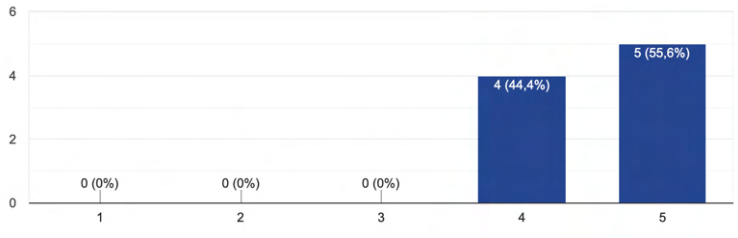

Figure 4.2: How useful is the tool.

For the majority of people interviewed, the tool is useful. This result is due to the fact that to the users it has been explained that HoloHelp has the goal to help people.

2. How accurate do you think object detection is? (*Figure 4.3*)

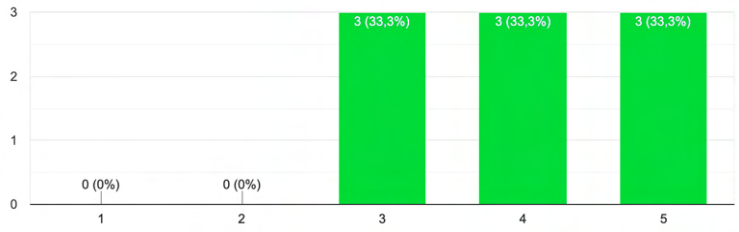

Figure 4.3: How accurate is object detection.

On average, the result is not so great. This is probably influenced by two factors: the model training (precision, recall, mAP) and the mapping from the 3D to the 2D world. Those are the two "weak links" of the tool.

3. Among the detected objects, which explanation do you believe to be the most accurate? (*Figure 4.4*)

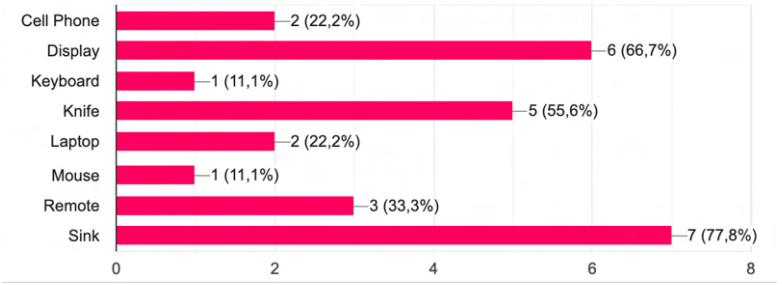

Figure 4.4: Which explanation is the most accurate.

Here we can see a remarkable interest of people interviewed in the explanation of the sink, which shows how to wash your hands step-by-step. This scenario is probably the most accurate because shows a useful explanation also for people to whom this tool is not referred directly.

4. How clear were the explanations? (*Figure 4.5*)

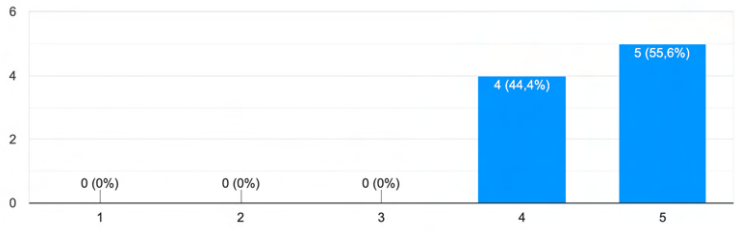

Figure 4.5: How clear were the explanations.

The overall vote is over 4 out of 5. This result clarifies that the story-telling is typically useful and clear to the users.

5. How useful did you find the audio feedback? (*Figure 4.6*)

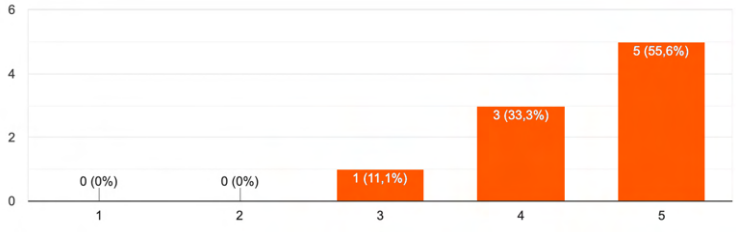

Figure 4.6: How useful is the audio feedback.

The audio feedback is, among all the kind of feedback, really useful. More than half considered it important, only one person found it not so useful.

6. How useful did you find the video feedback? (*Figure 4.7*)

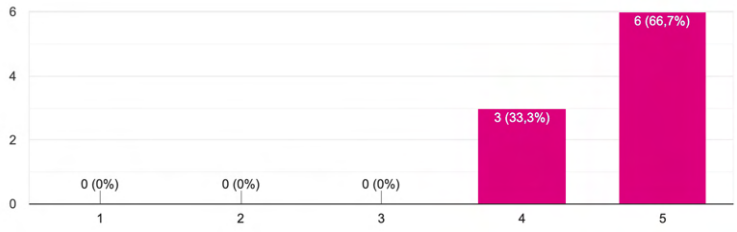

Figure 4.7: How useful is the video feedback.

The majority found interesting the video feedback. This is probably because this kind of feedback includes also the audio one.

7. How useful did you find the photo usage for the explanation? (*Figure 4.8*)

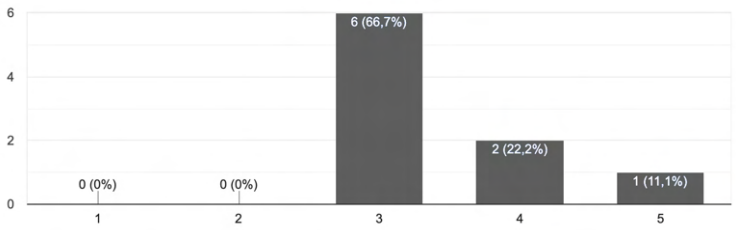

Figure 4.8: How useful is the feedback via photo.

On average, this is probably the least interesting feedback. This is due to the fact that this specific feedback consists of a concatenation of images and is not so explanatory.

Chapter 5

Conclusions

HoloHelp is created with the primary aim of helping people interacting with objects. During the thesis we have experimented different solutions, with the aim of making the tool as usable as possible for the majority of the users.

5.1 Usability

Along with the purpose of assisting users, particular attention has been paid to usability. It played a key role in the organisation of each part of the tool. In particular, for instance, at the beginning of the application it will be shown a **video and audio introduction** of what the tool does and what are its goals. Furthermore, **different kinds of feedback** are used also for the explanation of objects. In addition, if the prediction is not particularly reliable an **introductory audio message** will inform the user: "*I am not sure, but I think you would like to receive information about..*". Last but not least, the **eye tracking** it is also fundamental for the precision of the tool. Indeed, it was introduced also because of avoiding users to tap, move or select windows that might distract from the main role of the tool.

5.2 Possible Improvements and Extensions

There is an important number of improvements that could be made on HoloHelp. Here are some:

- **Prediction model upgrade**: as explained, the prediction model is not 100% trustworthy. This is due to the fact that an academic licence

has been used, so few images were uploaded. For the improvement of the model is sufficient increase the number of photo uploaded.

- **Delete saved images**: this is a code modification. At this moment, the tool saves the photos locally before they are sent. It delete those only at the end of each usage. This approach could cause an "overload" in terms of storage, so it might be improved using a "single-use" upload.
- **Offline prediction model**: Microsoft Azure allows to download the prediction of the model. This approach could led to an entire usage of the tool completely offline, eliminating the dependence on an internet connection.
- **Medical use**: the held skills do not provide the possibility to address this tool to specific categories. In a future moment, with the aid of a specialized team, the tool could be directly addressed to people with impairments, children, elders or labourers.

In conclusion, this experience has increased my technical skills and opened a world that was previously unknown to me. **HoloHelp** does not claim to be considered a medical or educational device but, as a prototype, **lays the foundations for a future way of learning differently**.

Appendices

.1 updateGaze function

This is the method called in the Update function. It is used to get eye gaze coordinates and make immediately after the mapping from the 3D to 2D world, in order to correctly do the prediction.

```
1  public void updateGaze()
2  {
3      Camera cam;
4      GameObject gazeObject;
5      Vector3 currentGazePos;
6      Vector2 viewPos = new Vector2(-1, -1);
7
8      if (gazeObject == null) { gazeObject = GameObject.Find("
           CursorRest"); }
9      try
10     {
11         if (gazeObject != null)
12         {
13             currentGazePos = Camera.main.transform.
                   InverseTransformPoint(gazeObject.transform.
                   position);
14             viewPos.x = focal_f * ((1.52048f * currentGazePos.
                   x) / currentGazePos.z);
15             viewPos.y = focal_f * ((1.52048f * currentGazePos.
                   y) / currentGazePos.z);
16
17             if ((viewPos.x < 0 || viewPos.x > cameraResolution
                   .width) || (viewPos.y < 0 || viewPos.x >
                   cameraResolution.height))
18             {
19                 currentGazePos = Camera.main.transform.
                       InverseTransformPoint(gazeObject.transform
                       .position);
20                 viewPos.x = focal_f * ((1.52048f *
                       currentGazePos.x) / currentGazePos.z);
21                 viewPos.y = focal_f * ((1.52048f *
                       currentGazePos.y) / currentGazePos.z);
22             }
23         }
24         else
25         {
26             currentGazePos = CoreServices.InputSystem.
                   EyeGazeProvider.GazeOrigin + CoreServices.
                   InputSystem.EyeGazeProvider.GazeDirection.
                   normalized * 1;
27             viewPos.x = focal_f * ((1.52048f * currentGazePos.
                   x) / currentGazePos.z);
```

```
28                viewPos.y = focal_f * ((1.52048f * currentGazePos.
                      y) / currentGazePos.z);
29            }
30            viewPos.x += (cameraResolution.width / 2);
31            viewPos.y = (cameraResolution.height / 2) - viewPos.y;
32        }
33        catch (InvalidCastException e)
34        {
35            Debug.Log(e);
36        }
37    }
```

Listing 1: updateGaze C# Script.

.2 Calibration

Microsoft HoloLens offers an ocular calibration system [12]. Because instructions are provided through holograms on HoloLens, it's very important that the holograms be correctly aligned (*see Figure 1*).

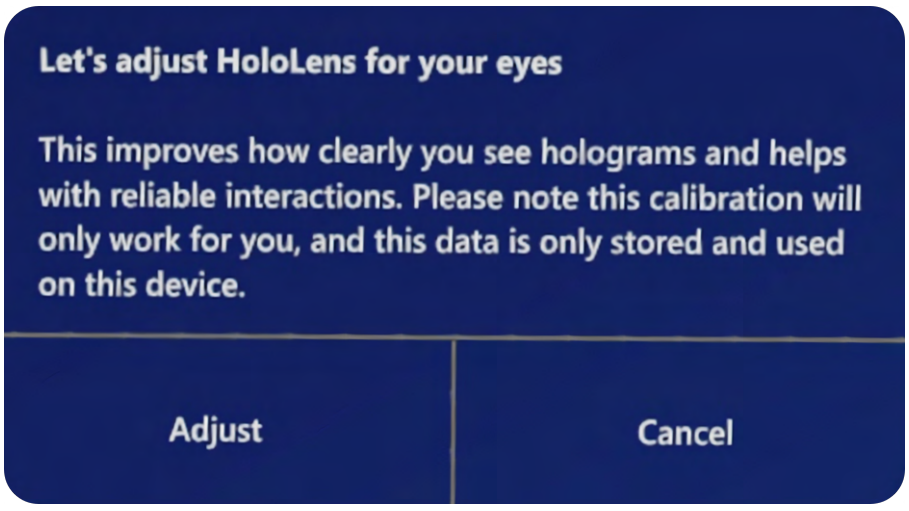

Figure 1: Example of a calibration pop-up in HoloLens.

.2.1 Use the Calibration app

1. If the pop-up doesn't appear, use the start gesture to open the **Start** menu.
2. If **Settings** isn't pinned to the **Start** menu, select the plus sign (+) button to view all apps.
3. Select **Settings**.
4. Select **System**.
5. In the side panel, select **Calibration**.
6. Select **Run Eye Calibration**.
7. Follow the on-screen instructions.

.3 Eye Tracking Offset

For a better usability, two different kinds of offset are used. The first, exploits a 50x50 range of pixels (*see Figure 2*).

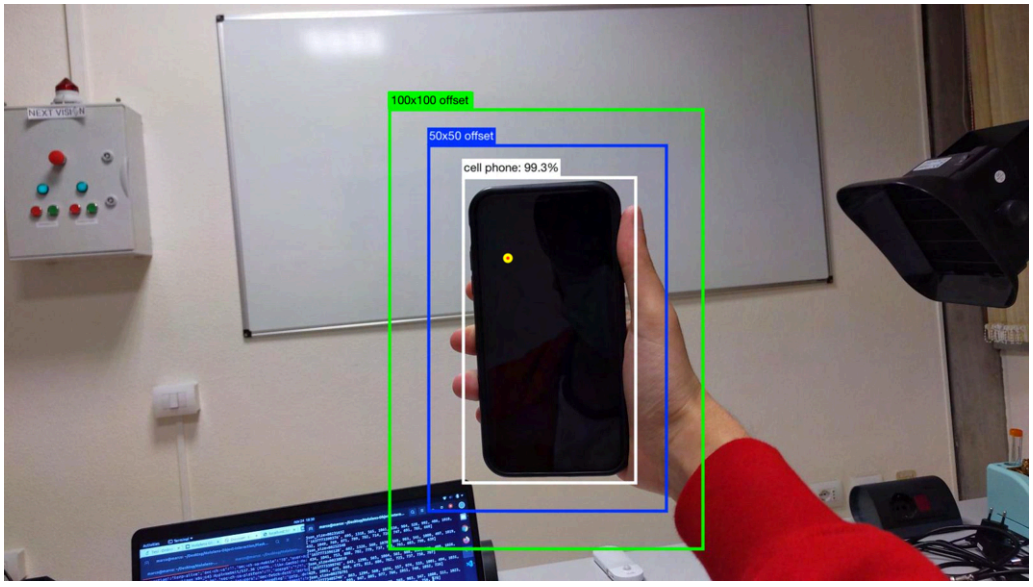

Figure 2: Example of 50x50 and 100x100 pixel offsets.

This means that for a better accuracy, if the gaze falls in that bounding box, it will be applied an extension of 50 pixels in height and 50 in width on it.

The same reasoning applies to the 100x100 range, with the difference that an audio feedback will inform the user on the uncertainty of prediction ("*I am not sure, but I think you would like to receive information about ...*").

```
// "Error" Pixels, 50x50 or 100x100 box enlarged
int offset = 50;

if (coordX >= (left - offset) && coordX <= (left + width +
    offset) && coordY >= (top - offset) && coordY <= (top +
    height + offset))
{
    tagName = p.TagName;
    probability = p.Probability;
    ...
}

int offset2 = 100;

else if (coordX >= (left - offset2) && coordX <= (left + width
    + offset2) && coordY >= (top - offset2) && coordY <= (top
    + height + offset2))
{
    // audio feedback: "I am not sure, ..."
    tagName = p.TagName;
    probability = p.Probability;
    ...
}
```

Listing 2: updateGaze C# Script.

Bibliography

[1] INAIL: Infortuni Malattie Professionali nel 2020
inail.it/../..infortuni-malattie-professionali-2020

[2] Wikipedia: Metaverse
en.wikipedia.org/wiki/Metaverse

[3] COCO: Common Objects in Context
cocodataset.org

[4] Wikipedia: Virtual Reality
en.wikipedia.org/wiki/Virtual_reality_headset

[5] Wikipedia: Object Detection
en.wikipedia.org/wiki/Object_detection

[6] Wikipedia: Unity (game engine)
en.wikipedia.org/wiki/Unity_(game_engine)

[7] Wikipedia: .NET Framework
en.wikipedia.org/wiki/.NET_Framework

[8] Microsoft: MRTK (Mixed Reality ToolKit)
docs.microsoft.com/en-us/windows/mixed-reality

[9] Microsoft Azure Cognitive Services
azure.microsoft.com/services/cognitive-services

[10] Microsoft Azure Custom Vision
customvision.ai

[11] COCO Link
cocodataset.org/#download

[12] Microsoft: Calibrate your HoloLens 2 device
docs.microsoft.com/.../operator-calibrate-hl2