NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

POSTGRADUATE PROGRAM
"INFORMATION AND DATA MANAGEMENT"

MSc THESIS

A Mixed Reality application for Object detection with audiovisual feedback through MS HoloLenses

Maria-Evangelia G. Pavlopoulou

Supervisors: **Stathes P. Hadjiefthymiades,** Professor
Nektarios N. Deligiannakis, PhD student

ATHENS

JULY 2022

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
"ΔΙΑΧΕΙΡΙΣΗ ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΔΕΔΟΜΕΝΩΝ"

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εφαρμογή Μεικτής Πραγματικότητας για αναγνώριση αντικειμένων με οπτικοακουστική ανατροφοδότηση μέσω MS HoloLenses

Μαρία-Ευαγγελία Γ. Παυλοπούλου

Επιβλέποντες: **Ευστάθιος Π. Χατζηευθυμιάδης,** Καθηγητής
Νεκτάριος Ν. Δεληγιαννάκης, Διδακτορικός φοιτητής

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2022

MSc THESIS

A Mixed Reality application for Object detection with audiovisual feedback through MS HoloLenses

Maria-Evangelia G. Pavlopoulou

S.N.: M1603

SUPERVISORS: **Stathes P. Hadjiefthymiades,** Professor
Nektarios N. Deligiannakis, PhD student

EXAMINATION COMMITTEE: **Lazaros Merakos,** Professor

July 2022

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εφαρμογή Μεικτής Πραγματικότητας για αναγνώριση αντικειμένων με οπτικοακουστική ανατροφοδότηση μέσω MS HoloLenses

Μαρία-Ευαγγελία Γ. Παυλοπούλου

Α.Μ.: Μ1603

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Ευστάθιος Π. Χατζηευθυμιάδης,** Καθηγητής
Νεκτάριος Ν. Δεληγιαννάκης, Διδακτορικός φοιτητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Λάζαρος Μεράκος,** Καθηγητής

Ιούλιος 2022

ABSTRACT

As computer science develops and progresses, new technologies emerge. Recent advances in augmented reality and artificial intelligence have caused these technologies to pioneer innovation and alteration in any field and industry. The fast-paced developments in computer vision and augmented reality facilitated analyzing and understanding the surrounding environments.

Mixed and Augmented Reality can greatly extend a user capabilities and experiences by bringing digital data directly into the physical world where and when it is most needed. Current smart glasses such as Microsoft HoloLens device excel at positioning within the physical environment, however object recognition is still relatively primitive. With an additional semantic understanding of the wearer's physical context, intelligent digital agents can assist workers in warehouses, factories, greenhouses, etc. or guide consumers through completion of physical tasks.

We present a mixed reality system that, using the sensors mounted on the Microsoft HoloLens headset and a cloud service, acquires and processes in real-time data to detect and track different kinds of objects and finally superimposes geographically coherent holographic tooltips and bounding boxes on the detected objects.

Such a goal has been achieved dealing with the intrinsic headset hardware limitations, by performing part of the overall computation in an edge/cloud environment. In particular, the heavier object detection algorithms, based on Deep Neural Networks (DNNs), are executed in a cloud RESTful system hosted by a server running on an NVIDIA Jetson TX2, a fast and power-efficient embedded AI computing device. We apply YOLOv3 (You Only Look Once) as a deep learning algorithm at server side to process the data from the user side, using a model trained on the public dataset MS COCO. This algorithm improves the speed of detection and provides accurate results with minimal background errors.

At the same time, we compensate for cloud transmission and computation latencies by running camera frames similarity check between the current and previous HoloLens camera capture frames, before applying the object detection algorithms on a camera frame, to avoid running the object detection task when the user surrounding environment is significantly similar and limit as much as possible complex computations.

This application also aims to use modern technology to help people with visual impairment or blindness. The user can issue a voice command to initiate an environment scan. Apart from visual feedback provided for the detected objects, the application can read out the name of each detected object along with its relative position in the user's view. A distance announcement of the detected objects is also derived using the HoloLens's spatial model. The wearable solution offers the opportunity to efficiently locate objects to support orientation without extensive training of the user.

SUBJECT AREA: Mixed Reality Applications with the HoloLens device

KEYWORDS: augmented reality, mixed reality, computer vision, object detection, HoloLens, YOLOv3, REST, NVIDIA Jetson TX2

ΠΕΡΙΛΗΨΗ

Καθώς η επιστήμη των υπολογιστών αναπτύσσεται και προοδεύει, εμφανίζονται νέες τεχνολογίες. Οι πρόσφατες εξελίξεις στην επαυξημένη πραγματικότητα και την τεχνητή νοημοσύνη έχουν κάνει αυτές τις τεχνολογίες να πρωτοπορήσουν στην καινοτομία και την αλλαγή σε κάθε τομέα και κλάδο. Οι ταχύρρυθμες εξελίξεις στην μηχανική όραση και την επαυξημένη πραγματικότητα διευκόλυναν την ανάλυση και την κατανόηση του περιβάλλοντος χώρου. Η μεικτή και επαυξημένη πραγματικότητα μπορεί να επεκτείνει σε μεγάλο βαθμό τις δυνατότητες και τις εμπειρίες ενός χρήστη, φέρνοντας ψηφιακά δεδομένα απευθείας στον φυσικό κόσμο όπου και όταν είναι απαραίτητα. Τα τρέχοντα έξυπνα γυαλιά, όπως η συσκευή Microsoft HoloLens, υπερέχουν στην τοποθέτηση εντός του φυσικού περιβάλλοντος, ωστόσο η αναγνώριση αντικειμένων εξακολουθεί να είναι σχετικά πρωτόγονη. Με μια πρόσθετη σημασιολογική κατανόηση του φυσικού πλαισίου του χρήστη, οι έξυπνοι ψηφιακοί πράκτορες μπορούν να βοηθήσουν τους χρήστες στην ολοκλήρωση εργασιών. Στην παρούσα εργασία, παρουσιάζεται ένα σύστημα μεικτής πραγματικότητας που, χρησιμοποιώντας τους αισθητήρες που είναι τοποθετημένοι στα HoloLens και μια υπηρεσία cloud, αποκτά και επεξεργάζεται δεδομένα σε πραγματικό χρόνο για την ανίχνευση διαφορετικών ειδών αντικειμένων και τοποθετεί γεωγραφικά συνεκτικά ολογράμματα που περικλείουν τα εντοπισμένα αντικείμενα και δίνουν πληροφορία για την κλάση στην οποία ανήκουν. Για την αντιμετώπιση των εγγενών περιορισμών υλικού των HoloLens, εκτελούμε μέρος του συνολικού υπολογισμού σε περιβάλλον cloud. Συγκεκριμένα, οι αλγόριθμοι ανίχνευσης αντικειμένων, που βασίζονται σε Deep Neural Networks (DNNs), εκτελούνται σε ένα σύστημα που υποστηρίζει RESTful κλήσεις δεδομένων και φιλοξενείται σε ένα NVIDIA Jetson TX2, μια γρήγορη και αποδοτική ενσωματωμένη υπολογιστική συσκευή AI. Εφαρμόζουμε το YOLOv3 (You Only Look Once) ως αλγόριθμο Βαθιάς Μηχανικής μάθησης, χρησιμοποιώντας ένα μοντέλο εκπαιδευμένο στο σύνολο δεδομένων MS COCO. Αυτός ο αλγόριθμος παρέχει ταχύτητα ανίχνευσης και ακριβή αποτελέσματα με ελάχιστα σφάλματα. Ταυτόχρονα, αντισταθμίζουμε τις καθυστερήσεις μετάδοσης και υπολογισμού εκτελώντας έλεγχο ομοιότητας μεταξύ των καρέ λήψης κάμερας των HoloLens, πριν εφαρμόσουμε σε ένα καρέ τους αλγόριθμους ανίχνευσης αντικειμένων, για να αποφύγουμε την εκτέλεση της εργασίας ανίχνευσης αντικειμένων όταν το περιβάλλον του χρήστη είναι αρκετά παρόμοιο και να περιορίσουμε τους πολύπλοκους υπολογισμούς. Αυτή η εφαρμογή στοχεύει επίσης στη χρήση σύγχρονης τεχνολογίας για να βοηθήσει άτομα με προβλήματα όρασης ή τύφλωση. Ο χρήστης μπορεί με φωνητική εντολή να ξεκινήσει μια σάρωση του περιβάλλοντος χώρου. Εκτός από την οπτική ανατροφοδότηση, η εφαρμογή μπορεί να διαβάσει το όνομα κάθε αντικειμένου που ανιχνεύτηκε μαζί με τη σχετική θέση του στον χώρο και την απόστασή του από τον χρήστη με βάση το χωρικό μοντέλο των HoloLens. Έτσι, υποστηρίζει τον προσανατολισμό του χρήστη χωρίς να απαιτείται εκτενή εκπαίδευση για την χρήση της.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Εφαρμογές Μεικτής Πραγματικότητας με τη συσκευή HoloLens

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: επαυξημένη πραγματικότητα, μεικτή πραγματικότητα, μηχανική όραση, εντοπισμός αντικειμένων, συσκευή HoloLens, YOLOv3, REST, NVIDIA Jetson TX2

To my family

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to my supervisor professor, Stathes P. Hadjiefthymiades, who gave me the chance to work on the field of Machine learning and Mixed reality and trusted me with this thesis. I would also like to extend my deepest gratitude to Nektarios Deligiannakis for his constant support and invaluable contribution throughout the duration of this project.

Special thanks to my family for their continuous support during my studies.

CONTENTS

LIST OF FIGURES

LIST OF IMAGES

LIST OF TABLES

PREFACE

This project has been developed in the National and Kapodistrian University of Athens at the Department of Informatics and Telecommunications as my postgraduate thesis. Hardware used was sponsored by the Pervasive Computing Research Group (p-comp) of the same department.

Athens, July 2022

1. INTRODUCTION

In this thesis we have created a Mixed Reality application deployed on the Microsoft HoloLens 1 device. The aim of the project is to provide a visual and audio description of

the elements that the user has in front of. The scenario that we consider is the one where a generic user has no familiarity or experience and wants to improve his knowledge on the surrounding environment. The goal is to make the application as much manageable and easy to use as possible. Using this application, the end user can select to initiate a visual or an audio-visual environment scan.

Upon selecting the visual scan menu option, the user can fire an environment scan with a simple hand tap gesture or with the simple voice command "Scan". Using the tools provided by Mixed Reality Toolkit [1], each time a scan is initiated, we acquire a camera capture of the user's view. This photo capture frame contains both the native image data and spatial matrices that indicate where the image was taken.

With respect to the intrinsic headset hardware limitations and with a main concern of building an application that is battery consumption friendly, the heavy process of applying the object detection algorithm on the camera capture frame taken is executed in a cloud RESTful system hosted by a server running on an NVIDIA Jetson TX2. NVIDIA Jetson TX2 is a power-efficient embedded AI computing device hosted on our research lab, a great choice as a backend system for our application, as it is capable of providing object detection results in a fast and reliable way. We apply YOLOv3 (You Only Look Once) as a deep learning algorithm at server side to process the data from the user side, using a model trained on the public COCO dataset. Once saved the photo capture frame from along with its spatial information, an API request will be sent to the object detection endpoint running on our server. A JSON containing the class label and the bounding box for each of the detected objects is returned to the HoloLens as a response.

The bounding box coordinates returned belong to the 2D image coordinate system and are associated with the camera frame picture taken. Since the depth of a point can be calculated by associating it with an environment map provided from HoloLens API, the system can estimate where the point on the image is in the world coordinates. With the world coordinates for each of the four vertices of each bounding box in our hands, we can draw geographically coherent holographic tooltips and bounding boxes on the detected objects.

When the user selects the audio scan menu option, the environment scan process starts automatically and runs periodically. We compensate for data transmission and computation latencies by running camera frames similarity check between the current and previous HoloLens camera capture frames, a functionality available in another endpoint running on our NVIDIA Jetson TX2 server. This is done before applying the object detection algorithms on a camera frame, to avoid running this heavy task when the user surrounding environment is significantly similar and limit as much as possible complex computations. For audio scan, apart from visual feedback provided for the detected objects, the application can read out the name of each detected object along with its relative position in the user's view. Using the HoloLens's spatial model, we can also derive and announce distance information for each of the detected objects from the user.

Many trials and fails had to happen in order to develop and create a stable and reliable version of the above application. With its modular design, this proof-of-concept implementation serves as a blueprint for future applications and is a first step towards the successful integration of Machine Learning based Mixed Reality systems into industrial quality inspection settings.

2. STATE OF THE ART

2.1 Introduction

As technology has advanced, so the way of visualizing simulations and information is changed in the digital era. Virtual Reality, Augmented Reality and Mixed Reality are great examples of such visualization methods which are booming in this period, either by being immersed in a simulated virtual environment or adding a new dimension of interaction between digital devices and the real world. These methods have something similar, equally significant in their ways providing experiences and interaction being detached or blending with the real world, making real and virtual alike. The process of replacing and supplementing the real world according to the needs is what makes these methods more desirable and increasingly popular. From consumer application to manufacturers these technologies are used in different sectors, such as gaming, education, defense, tourism, aerospace, corporate productivity, enterprise applications, and so on, providing huge benefits through several applications, even if has to be said that they are still immature technologies, in a state of evolution.

Although these technologies look similar in the way they are used, and sometimes the difference is confusing to understand, there is a very clear boundary that distinguishes these technologies from each other. Virtual Reality henceforth VR, is the most widely known of these technologies. It is fully immersive, which tricks the users' senses into thinking they are in a different environment or world apart from the real world. Using a head-mounted display (HMD) or headset, we experience a computer-generated world of imagery and sounds in which we can manipulate objects and move around using haptic controllers while tethered to a console or PC. Augmented Reality, or simpler AR, overlays digital information on real-world elements. Augmented reality keeps the real world central but enhances it with other digital details, layering new strata of perception, and supplementing the learner’s reality or environment.

Mixed Reality or MR brings together real world and digital elements. In mixed reality, we interact with and manipulate both physical and virtual items and environments, using next generation sensing and imaging technologies. Mixed Reality allows us to see and immerse ourselves in the world around us even as we interact with a virtual environment using our own hands—all without ever removing the headset. It provides the ability to have one foot (or hand) in the real world, and the other in an imaginary place, breaking down basic concepts between real and imaginary, offering an experience that can change the way we game and work today.

At this point the community differentiates the MR and AR based on the devices that are used for the applications. AR is referred to the experience offered from a mobile device application and MR is a term used to refer to applications that are deployed on wearable devices, mostly with a head-mounted display.

We use the HoloLens device created and supported from Microsoft as a reference and as our targeted device for our application. Microsoft HoloLens were introduced in 2016 as a cutting-edge technology device, supporting Mixed Reality applications. HoloLens maps the 3D space around the user and projects holograms in front of his eyes, allowing him to interact with them in a natural way. We provide a detailed presentation of this device in a following chapter.

The purpose of this chapter is to examine in detail these state-of-the-art technologies, devices related to them and applications.

2.2 Virtual Reality

Virtual reality means entirely replacing the reality you see around you with computer-generated 3D content. With head-mounted displays for VR (HMD), also known as VR headsets, you're completely immersed in the virtual simulation and cut off from the real world. This allows you to freely customize and create as many different experiences and scenarios as needed and, depending on the device, you can interact with virtual contents and objects using your hands or with your eyes.

HMDs basically consists of stereoscopic displays and motion tracking hardware. The way HMDs implement stereoscopic displays is by generating different image for each eye, which results in generating the illusion of depth. Within HMDs, motion tracking hardware mostly consists of a gyroscope and accelerometer to measure motion/position changes. This helps in simulating real-world experiences.

VR headsets can be tethered to a PC, enabling a more powerful graphics. Alternatively, VR headsets can be untethered, which lowers the visual quality but allows the user to move around freely without wires. The virtual experience that you see can be anything between a photogrammetric capture of the real reality or a computer-generated scene that is 3D modeled and built with a gaming engine. Most VR headsets are connected to a computer (Oculus Rift) or a gaming console (PlayStation VR) but there are standalone devices (Google Cardboard is among the most popular) as well. Most standalone VR headsets work in combination with smartphones.

Image 1: The Virtual Reality Experience [9]

While VR headsets can provide fairly immersive experiences, they also place limitations on the space where the headset is used as the user's ability to interact with real-world objects is limited. Collaboration with your (physical) colleagues is also limited as communicating with them and using devices in the same shared space are difficult.

Invented in the 1950s, VR's development has experienced peaks and troughs. The first VR HMD system, The Sword of Damocles, was invented in 1968 by computer scientist Ivan Sutherland and his student Bob Sproull. Meanwhile, the term "virtual reality" was popularized by Jaron Lanier in the 1980s. Ten years later, VR was used for training and simulation in the US military and the National Aeronautics and Space Administration (NASA). Mass production of VR systems began in the early 1990s, led by Virtuality, which opened dedicated VR arcades. Contemporary VR devices emerged with the introduction of the PC-connected Oculus Rift prototype in 2010. Between 2014 and 2017, the market progressed from PC-tethered headsets (the HTC Vive) to console-tethered headsets (Sony's PSVR) and mobile-tethered headsets (Samsung GearVR and Google Cardboard). Untethered headsets (Oculus Go, Lenovo Mirage Solo, and HTC Vive Focus) arrived in 2018, making VR an independent platform [7].

There are various implementations of VR in diverse fields currently, and numerous new ones are coming up every day. Some of the current fields where VR is being implemented include gaming, tourism, education, architecture, enterprise productivity and web content. For instance, Space Explorers is a VR app that presents you with real

International Space Station footage. Claiming to be the largest production ever filmed in space, this four-part series was shot over a period of two years and allows the user to mingle with the astronauts aboard the ISS [8].

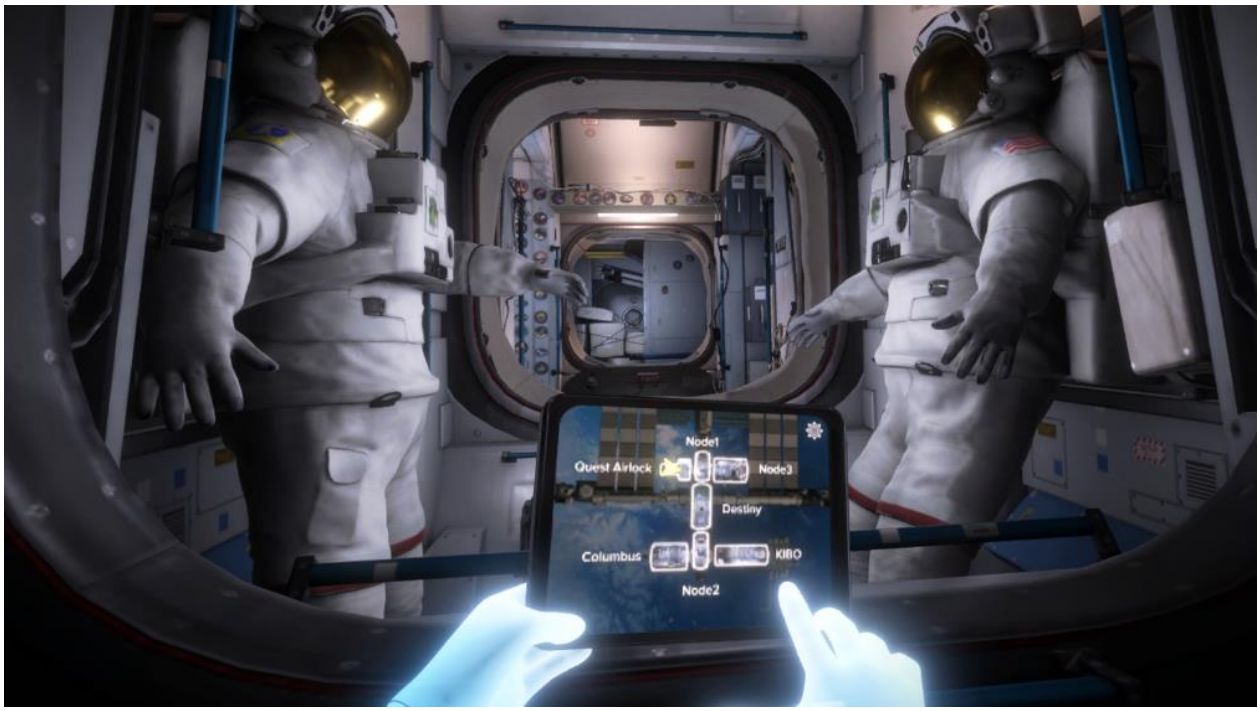


Image 2: The Oculus ISS Experience (OCULUS VR, LLC) [10]

2.3 Augmented Reality

AR is all about bringing digital information and overlaying it over the real environment. The only difference from VR is that it creates a totally artificial environment around you, whereas AR uses the environment around you and overlays the digital information over it. For VR, you will require a VR device, but for AR it can be achieved by simply using a smartphone, tablet, or dedicated VR devices. AR can be categorized into three different types based on the display types:

- Screen-based
- HMD-based
- Projection-based

2.3.1 Screen-based displays

A common example of screen-based AR is overlaying digital information over smartphones or tablet camera displays. For example, you switch on and point your smartphone/tablet camera over an object, and the application recognizes that object

and overlays that object information, such as the price or description, as digital information over the object image. Another example of screen-based AR is the video game Pokémon GO by Niantic which became hugely popular with players of all ages from all over the world. In this game, based on the user's location and direction, digital characters are overlaid over video images.

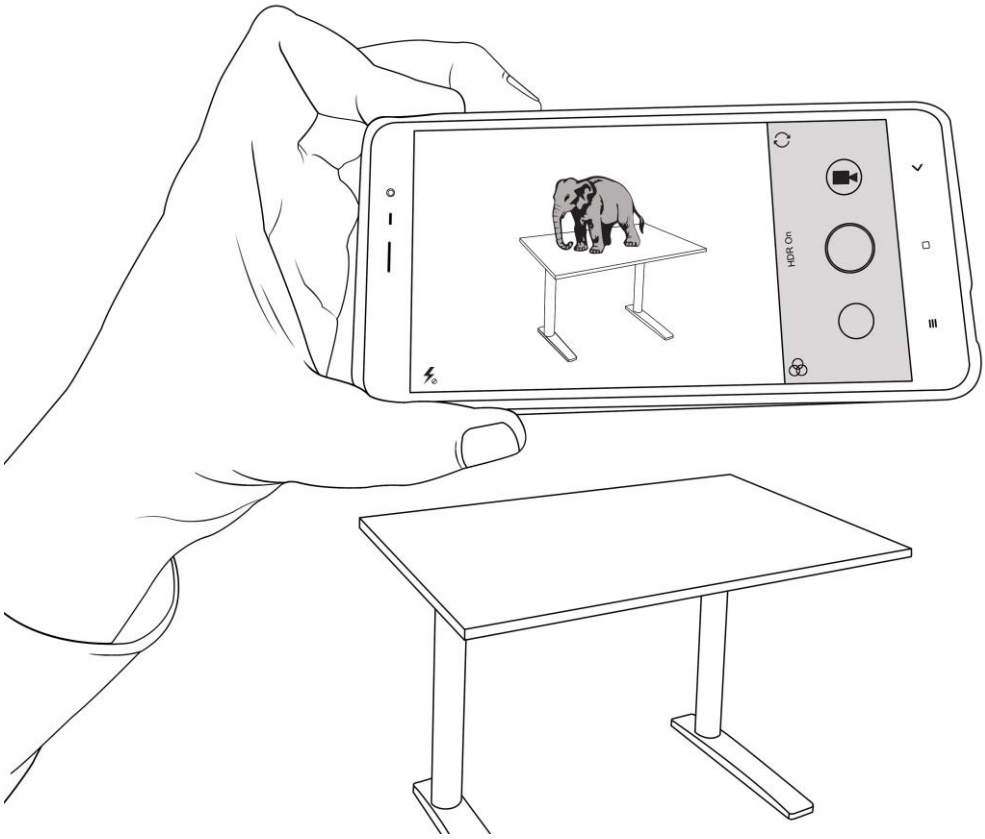


Figure 1: User is viewing augmented object using smartphone [6]

In the preceding figure, a user is viewing an augmented object, that is, the elephant's digital object is overlaid over the video frame. This is an example of screen-based AR.

2.3.2 Head-mounted globally displays

Using AR HMD devices, digital information is overlaid directly over the user's view of the real world. So, there is no need to hold any screen to view the digital information. Digital information is directly rendered over the view area of the user's eyes. In the following figure, the user is using a head mounted AR device to view the physical element in front of him. Within the view of the user, the AR device embeds information about the physical object and provides them with a more immersive experience.

An example of AR headset is Google Glass, where digital content is displayed on a tiny screen in front of a user's eye.

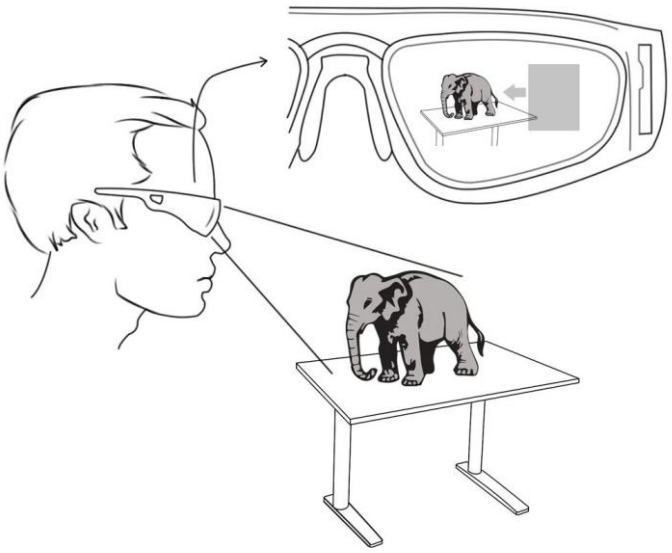


Figure 2: HMD with information overlaid over the real view [6]

2.3.3 Projection-based displays

In user projection-based AR, a projection is rendered on the target surface itself. This target surface could be anything, such as building, person, room, and so on. To render this kind of projection, the system needs to know the exact dimensions of the target surface, and then using single/multiple projectors, it renders the projection on the target.

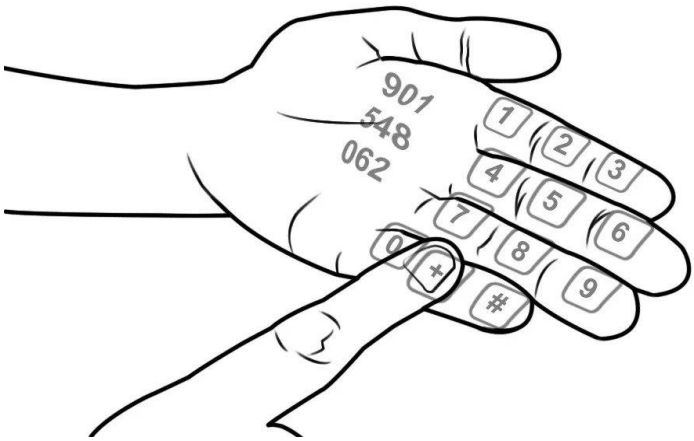


Figure 3: Phone dialer pad projection over the hand [6]

In the preceding figure, the user is viewing a projection of the phone dialer on the palm of their hand.

Applying AR is quite different from applying VR. AR applications are more focused on integration scenarios with the real world. In e-commerce applications, businesses can't reach each customer with demo-able physical product, and opening showrooms is every city is a costly business, especially for start-ups and newcomers in the market. So, reaching out to customers through AR, online furniture retail companies, for example, allows users to consume AR through smartphones, let them visualize their furniture products, and enable them to design their house interiors. Such an example is IKEA Place AR application.

Also, we can find many use case scenarios in military applications. An application we recently worked on was the Naval Fire Fighting Training and Education System (NAFTES) [14], a project that was implementing some training exercises regarding firefighting safety and training for the Hellenic Military Navy. This application aimed to introduce to the Navy a new technology to present the AR technology to train sailors when at sea.

To conclude, the possibilities of AR tech are limitless. The only uncertainty is how smoothly, and quickly, developers will integrate these capabilities into devices that we'll use on a daily basis, which yet has many difficulties.

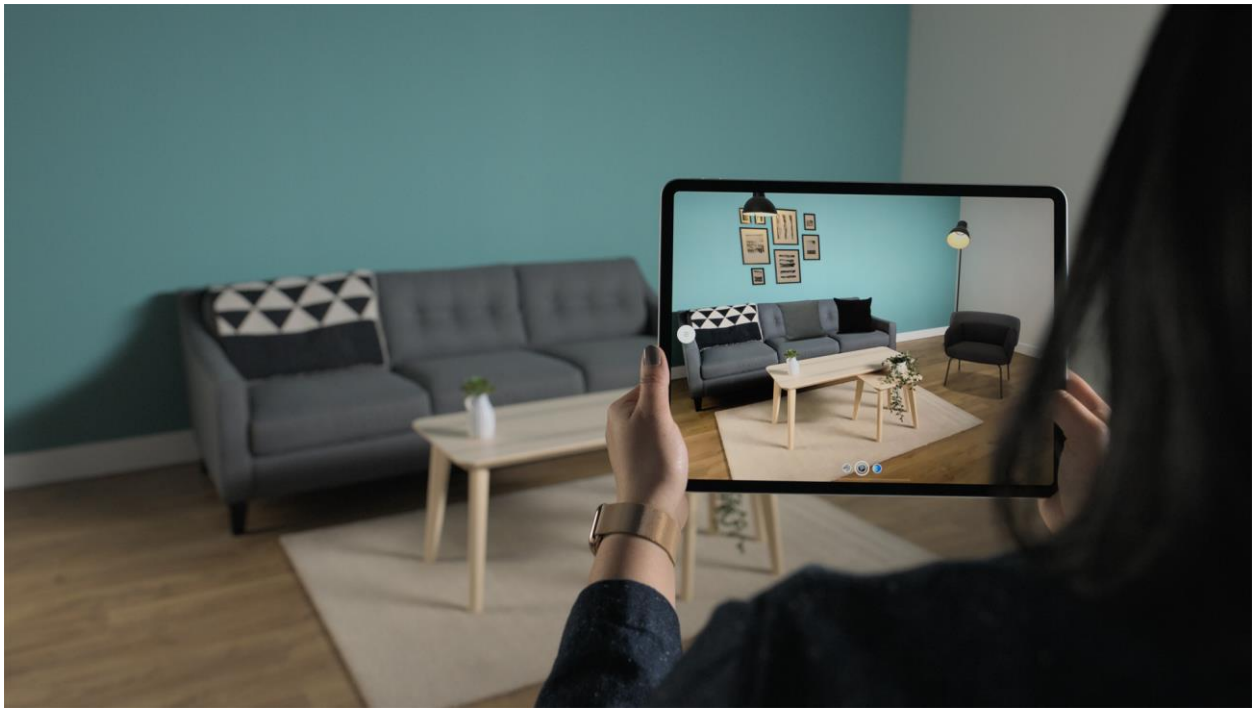


Image 3: Ikea Place Augmented Reality App [13]

2.4 Mixed Reality

Mixed reality (MR) is a spectrum of immersive experiences, connecting and blending physical and digital worlds together in augmented reality and virtual reality applications.

In visual terms, imagine mixed reality as a creative space that exists between the extremes of the physical and digital worlds. Experiences range from overlaying virtual content on objects in the physical world, like in augmented reality apps, to an entirely immersive experience where the user doesn't have any input from the real world, as in virtual reality.

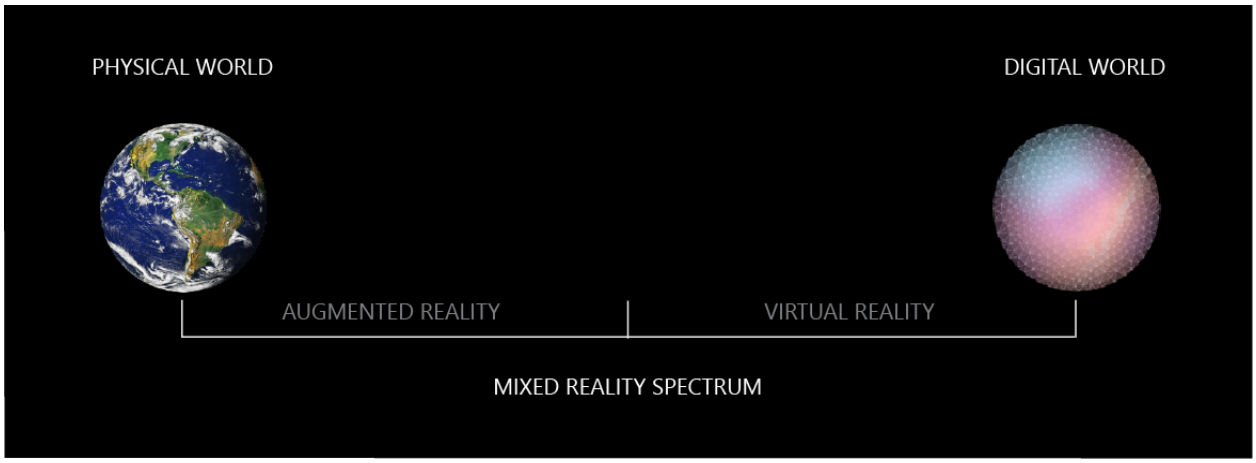


Figure 4: Milgram's Reality-Virtuality Continuum [18]

Because mixed reality covers such a broad range of possible user experiences, it comes with a set of interaction types that are entirely unique. These interaction types include but are not limited to:

- Environmental input, like capturing a user's position in the world by mapping surfaces and boundaries in the area.
- Spatialized sound, which is 3D sound that has position and depth in a virtual space, just like in the real world.
- Locations, positions, and persistence of objects in real and virtual spaces.

These features are part of a relationship between human and computer input known as human-computer interaction (HCI). Human input covers the more familiar ways of interacting with technology, such as using a keyboard, mouse, touchpad, or your own voice. As sensors and processing power have increased in computers, so has this new area of computer input from the environment. The interaction between computers and environment is called perception.

The intersection of computer processing, human input, and environmental understanding is where the power, creativity, and emerging capabilities of mixed reality come to life. Movement through the physical world can translate to movement in the digital world. Boundaries in the physical world can influence application experiences, such as game play, in the digital world. Without environmental input, experiences can't blend between physical and digital realities.

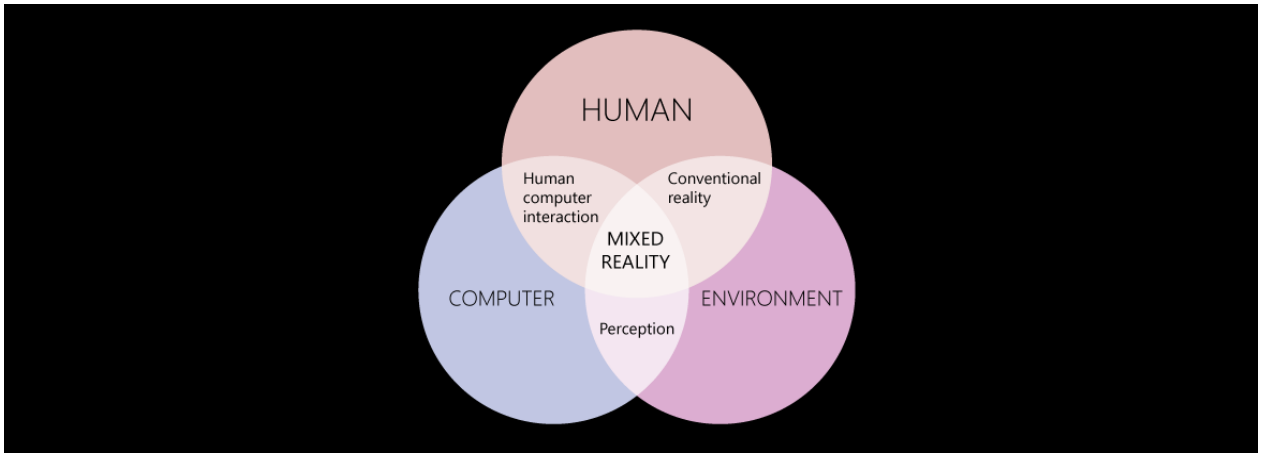


Figure 5: Mixed Reality Venn Diagram [18]

As a historical footnote, the term mixed reality comes from a 1994 paper titled “A Taxonomy of Mixed Reality Visual Displays”, written by Paul Milgram and Fumio Kishino [17]. Although that paper wasn't describing the version of mixed reality that we have today, it did explore the concept of a virtuality continuum and the categorization of taxonomy applied to displays, which is depicted in Figure 4.

MR combines the best aspects of both VR and AR. It is all about merging virtual content with the real world in an interactive, immersive way. Virtual objects appear as a natural part of the real world, occluding behind real objects. Real objects can also influence the shadows and lights of virtual contents. This natural interaction between real and virtual opens up a whole new realm of solutions that would not be possible with virtual or augmented reality.

MR gives the ability to see yourself and interact with your colleagues while, for example, designing a virtual object or environment. For MR to be valuable for professionals, it has to be convincing – blending real and virtual content to the point that it’s impossible to tell where reality ends and the virtual world begins. MR is best accomplished with video pass-through technology instead of optical see-through. With video pass-through based solutions, virtual objects can be black or opaque, and appear as solid as anything in the

real world. Colors are perfectly rendered, appear just as they should and you can also add, omit and adjust colors, shadows and light in the virtual world and the real world.

All of this means that we need fairly powerful computer hardware to run these experiences. However, recent advancements in the technology have made it achievable with high-end consumer desktops so any business now has the capacity to adopt mixed reality solutions, and the business impact can be astounding.

Microsoft (the company which device we used for our application) was one of the first to introduce to the public a HMD device, the HoloLens 1 that came with some built in applications and holograms. Although the device initially was very expensive, it started many discussions and research topics. At his point of time the community of developers seems to grow and new devices are announced or produced like the HoloLens 2 or the Apple AR Glasses.

Image 4: The Mixed reality Experience using Microsoft HoloLenses [19]

MR is still evolving, and lot of different industries are still exploring possibilities with it. The following are some of them, which represent early progress:

- Education and training: MR has already started transforming the way education and industrial training are delivered. Companies have started using MR as a medium for delivering readiness/training content to participants.

- Exploring new markets: MR is giving an opportunity to enterprises to rethink their marketing strategy and target new customers. Few popular automobile companies have started using MR device in their specialized car showrooms. With this, they can target consumers early in the sales cycle, even before the first model of the car is available.
- Healthcare: The possibilities for MR in the healthcare industry are enormous. Physicians can use MR devices to visualize consolidated reports, and these devices can also be used as a guidance device during operations or diagnosis.
- Gaming: The gaming industry may become one of the biggest consumers of MR devices. There are so many possibilities for MR games that can interact with the real world.

2.5 Differences between Virtual Reality, Augmented Reality and Mixed Reality

Virtual Reality and Augmented Reality are two sides of the same coin. As depicted in Figure 4, we could think of Augmented Reality as VR being closer to the real world: Augmented Reality simulates artificial objects in the real environment, while Virtual Reality creates an artificial environment to immerse into.

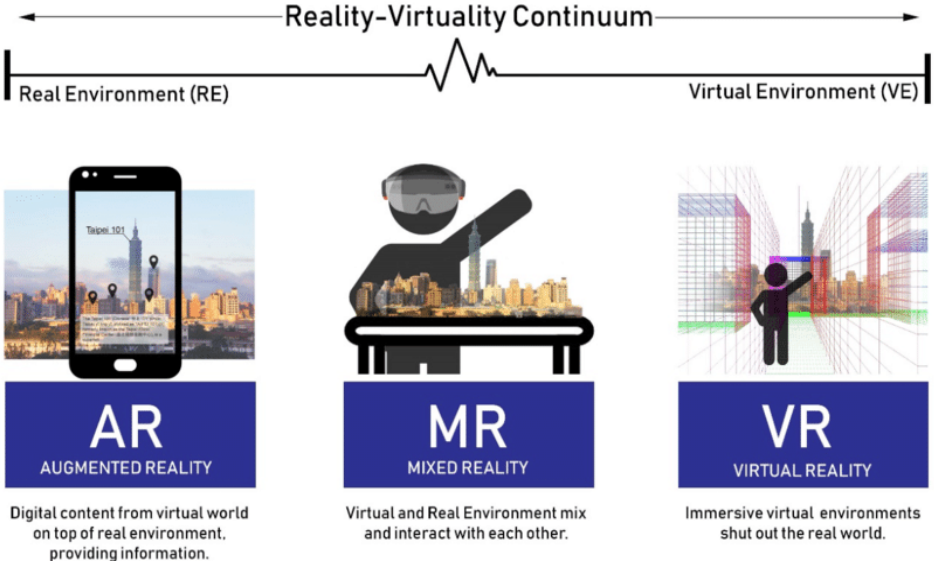


Figure 6: Differences between AR, MR and VR [21]

There are numerous distinctions between virtual reality and augmented reality. One of them is that virtual reality completely isolates the user from the actual world by allowing

him to experience and view a wholly digital environment (through VR equipment). In Augmented Reality, the user can view digital content embedded in the actual world keeping an eye on the real world. Computer vision algorithms and sensors are used in AR applications to recognize various aspects of the real world, such as surfaces and walls. These attributes are taken into account while rendering 3D objects.

Microsoft HoloLens goes beyond augmented and virtual reality by allowing us to interact with three-dimensional holograms that are seamlessly integrated into our surroundings. It is more than just a heads-up display, and its transparency ensures that we never lose sight of our surroundings. Integrating high-definition holograms with the real-world environment will open up entirely new ways to create, connect, work, and play.

Table 1: Differences between VR, AR and MR

	Augmented Reality	**Mixed Reality**	**Virtual Reality**
Real world augmentation	✓	✓	
Interactive Holograms		✓	
Virtual world experience		✓	✓
Replaces real world			✓

2.6 Digital Reality and the current marketplace

Since we learned about VR, AR, and MR, we can see what devices are currently available on the market to explore all these different experiences:

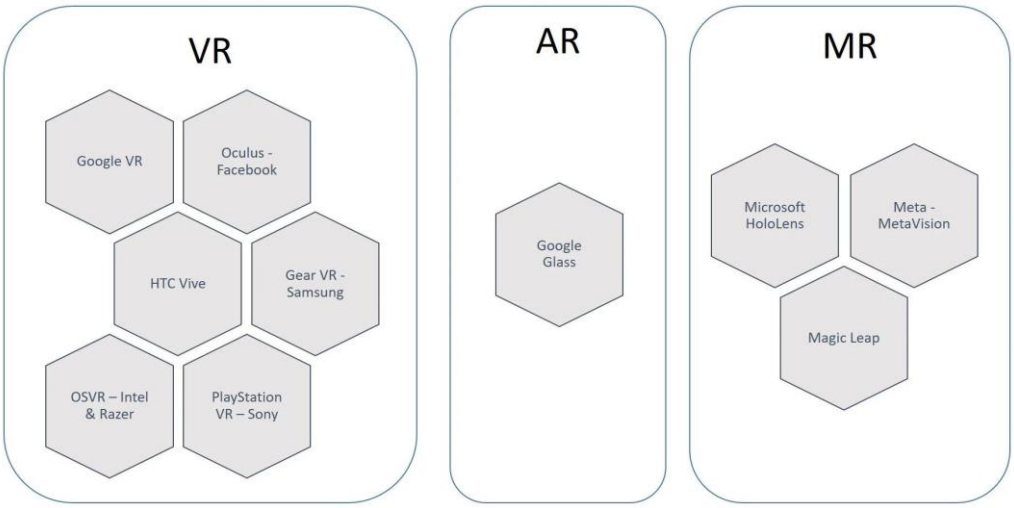


Figure 7: Digital Reality device family [6]

Digital Reality device family includes:

- Oculus: Owned by Facebook, the first in the industry to launch a VR device
- HTC Vive: A VR device launched by the Taiwanese phone manufacturer HTC
- Sony PlayStation VR: A VR device that works with Sony PlayStation, but also works with any personal computer
- Google VR: This is a Google VR device, and is used along with Android smartphones
- Gear VR: This is a Samsung VR device, and is used along with Galaxy smartphones
- OSVR: An open-source VR device, started with the support of companies such as Intel and Razer
- Google Glass: An AR device launched by Google
- Met: Mixed or Augmented Reality device, which projects virtual images in the wearer's field of vision
- Magic Leap: MR device, still under development
- HoloLens: Microsoft MR wireless and wearable device, used for our application, described in detail in the following chapter

2.7 HoloLens Devices

Microsoft HoloLens are a pair of mixed reality smart glasses developed and manufactured by Microsoft. As of today, there are two versions, HoloLens 1st Gen and HoloLens 2 (or HoloLens 2nd Gen).

2.7.1 HoloLens 1st Gen

The Microsoft HoloLens (1st generation) [22] is the world's first completely untethered holographic computer. HoloLens combines cutting-edge optics and sensors to produce 3D holograms that are anchored to the real world around you. The Microsoft HoloLens was the first head-mounted display to run the Windows Mixed Reality platform under the Windows 10 operating system. The tracking technology in HoloLens can be traced back to Kinect.

Image 5: HoloLens 1st generation [22]

The device capabilities that support the MR functionality are:

Using the following to understand user actions:

- Gaze tracking
- Gesture input
- Voice support

Using the following to understand the environment:

- Spatial sound

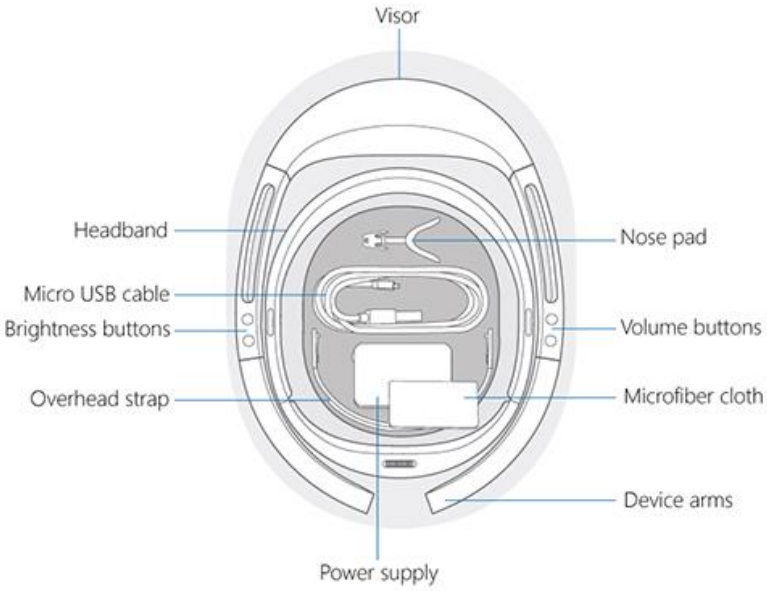


Figure 8: HoloLens components [22]

As of 2016, a number of augmented-reality applications have been announced or showcased for Microsoft HoloLens. A collection of applications will be provided for free for developers purchasing the Microsoft HoloLens Developer Edition. Applications available at launch include:

- Cortana, Microsoft's virtual assistant.
- Holograms, a catalog of a variety of 3D objects that users can place and scale around them, ranging from tigers and cats to space shuttles and planets.
- HoloStudio, a full-scale 3D modeling application by Microsoft with 3D print compatibility.
- CAE VimedixAR is a commercial application of Microsoft HoloLens technology that enables immersive simulation-based training in ultrasound and anatomical education through augmented reality for increased patient safety and enhanced learning.
- An implementation of the Skype telecommunications application by Microsoft. Any user with Skype on his or her regular devices like PC, Mobile etc. can dial user on HoloLens and communicate with each other. With Video Call On, the user on PC will see the view HoloLens user is seeing and HoloLens user will see view captured by PC / Mobile device user camera.
- HoloTour, an audiovisual three-dimensional virtual tourism application developer by Microsoft and Asobo Studio.
- Fragments, a high-tech crime thriller adventure game developed by Microsoft and Asobo Studio, in which the player engages in crime-solving.
- Young Conker, a platform game developed by Microsoft and Asobo Studio, featuring a young version of Conker the Squirrel.

- RoboRaid (previously code-named "Project X-Ray"), an augmented-reality first-person shooter game by Microsoft in which the player defends against a robot invasion, aiming the weapon via gaze, and shooting via the Clicker button or an air tap.
- Actiongram, an application for staging and recording short video clips of simple mixed-reality presentations using pre-made 3D virtual assets, will be released in summer 2016 in the United States and Canada.

Microsoft announced in November 2018 that it is preparing HoloLens for combat. The company received a $480 million military contract from the US government to incorporate AR headset technology into the arsenal of American soldiers.

2.7.2 HoloLens 2nd Gen

The second generation arrived to address some of the issues raised by the first. The device's components and operation remained mostly unchanged, however there were some significant enhancements. The HoloLens 2 MR headset has several notable enhancements, including:

- Processing power: with its Snapdragon 850 Compute Platform, the HoloLens 2 is more powerful than its predecessor.
- Field of view (FOV): at 52°, the HoloLens 2's field of view is larger, offering a more immersive MR experience for the user. The original HoloLens only has an FOV of 30°.
- Battery life: The HoloLens 2 features a 3-hour battery life, while the HoloLens 1 has a 2.5-hour battery life.
- Design and fit: according to Microsoft, the HoloLens 2 offers a lighter and more ergonomic fit than the original HoloLens. The HMD also now features a flip-up visor which allows users to enter/exit mixed reality more quickly [24].

Image 6: HoloLens 2 [23]

2.8 Machine Learning based Augmented Reality

One key area that Mixed Reality is really taking a strong foothold is in the field of Machine Learning (ML) and specifically of computer vision. With the use of simultaneous localization and mapping, we get a computer vision algorithm that compares visual features between camera frames in order to map and track the environment. When combined with sensor data from available devices such as gyroscope and accelerometer, it is possible to the precision of the tracking data. The purpose of machine learning is to bring context into the world of MR. We take real-world items and scenarios, add context using MR with customized experiences that are a result of Machine Learning.

The real advantage of Machine Learning in the world of business is using context to answer customer inquiries, quell fears and concerns, and field different scenarios. In a nutshell, Machine Learning makes use of a large subset of data in conjunction with MR to bring meaning to what users perceive and view.

There is a strong market drive to get the technology developed as soon as possible. That is why companies such as Apple, Google and others are restlessly working to improve these algorithms. This leaves developers with the task to continuously build better and more reliable applications using MR.

3. TOOLS FOR THIS APPLICATION

We used specific tools, APIs, and libraries to develop and deploy our application. We used Unity Engine in conjunction with the Unity Editor, as well as Visual Studio for the.NET Framework. We also used Microsoft's Mixed Reality Toolkit, which is currently at version 2.8.0.

Unity Editor is used to create games, virtual reality, and augmented reality applications. In our case, we have a mixed reality application that is somewhere between virtual reality and augmented reality.

Microsoft's Mixed Reality Toolkit is an open-source project designed and developed for virtual reality headsets or HoloLens devices.

To deploy the application to the HoloLens device, we have to install and run Visual Studio 2019, which enables the deployment of applications for Microsoft devices. The .NET framework for C# can be installed using the above tool.

We also used Python to write our scripts running on NVIDIA Jetson TX2 used to create a REST API for applying object detection on a camera frame from HoloLens device to get the detected objects and obtain a similarity score between two camera frames.

3.1 Unity Editor 2019 LTS and 2020

Unity Editor [25] is a tool for developing apps for various platforms. We had to use this tool to develop our application for the HoloLens device, because it was the only one that supported the libraries we needed to use with our HoloLens device. As a result, opting for this tool was something we needed to deploy our application to our HoloLens device. For instance, the MRTK from Microsoft was created to be loaded in Unity. This limitation did not prevent us from creating our application.

Unity Engine is a tool that is constantly being updated, and this has proven useful many times to fix bugs and problems that have decelerated the development of the application. However, it was possible that these updates could cause serious issues, forcing us to make changes to our project with no guaranteed outcome. Consequently, the correct configuration is critical for the application to work on different versions of Unity.

At first, we have to install a version of Unity. We recommend using an LTS version of Unity because choosing an unstable version is probable to provoke many difficulties

and problems. We first chose to use version 2020 of Unity, but since at the time we started the development of this thesis this version was not stable, Unity 2019.4.34f1 was finally chosen. However, new versions are already available, though some configuration may be required to successfully load and deploy the project.

The selected version is also associated with the MRTK version. Each version suggests the best Unity version that can be used to avoid any issues. It should be noted that this library was created exclusively for the Unity platform.

In the following sections, we will go over how to configure the Unity editor and other components so that we can edit and deploy our app to any device.

3.2 Mixed Reality Toolkit

Mixed Reality Toolkit [1] is a Microsoft open-source project that provides numerous functionalities for various devices and platforms. It also supports HoloLens 1 and is a very useful tool with many functions for handling user input actions. It is a package containing all of the components required to apply all of the mixed reality experience to our application.

The MRTK consists of 5 Unity asset packages, which are provided by the official Microsoft repository at GitHub [28]. These are:

- Foundation: The Mixed Reality Toolkit Foundation is the set of code that enables your application to leverage common functionality across Mixed Reality Platforms.
- Extensions: This optional package includes additional services that extend the functionality of the Microsoft Mixed Reality Toolkit. These additional services are about hand physics and scenes transitioning and more.
- Tools: This optional package includes helpful tools that enhance the mixed reality development experience using the Microsoft Mixed Reality Toolkit.
- Test utilities: This optional test utilities package contains a collection of helper scripts that enable developers to easily create play mode tests. These utilities are especially useful for developers creating MRTK components.
- Examples: The examples package is structured to allow developers to import only the examples of interest.

For our project we only used the required Foundation package.

Prior to the creation of the MRKT, another toolkit, known as the HoloToolkit, was only available for the HoloLens 1 device. When the HoloLens 2 was announced, an open source MRKT based on the HoloToolkit but with cross-platform aspects and many features for the HoloLens 2 device was released. There are dedicated guides available for migrating a project that uses the older HoloToolkit to the newest MRTK [26].

The main features that are offered are listed below:

- Input system for the various devices.
- Hand tracking features that are not currently used in our project
- Profiles for the various devices. These profiles are collections of functions that handle specific behaviors (in the sense of functionalities) for each device. In our case, we used the Default profile, which contains very basic functionalities for our device's mixed reality experience.
- UI Controls
- Solvers are scripts that implement automations primarily for an application's UI elements. I our case, we used the Follow Me Toggle with its solvers to make our

  UI elements follow the user’s position and view.
- Spatial Awareness and many other feature areas that we will mention in the following pages.
- Diagnostics tools

As we can see, there are many features implemented within the MRTK that aid in the development of useful applications that incorporate a great number of features.

From all of the above, we must highlight the Diagnostics system, which is critical in understanding what the application is doing and what overhead or cost on resources we have at any given time. Metrics can display memory status and utilization, CPU and GPU utilization, as well as frame statistics.

As depicted in Image 7, that grey box that follows us around in the project’s simulated scenes is called Visual profiler. This is a diagnostic tool which gives real-time information about the current FPS and memory usage in application view. The Visual Profiler can be configured via the Diagnostics System Settings under the MRTK Profiles Inspector. It is important to always keep track of the frame rate and the memory usage

and meet the target framerate, as outlined by the platform being targeted (i.e., Windows Mixed Reality, Oculus, etc.). For example, on HoloLens, the target framerate is 60 FPS. Low framerate applications can result in deteriorated user experiences such as worsened hologram stabilization, world tracking, hand tracking, and more.

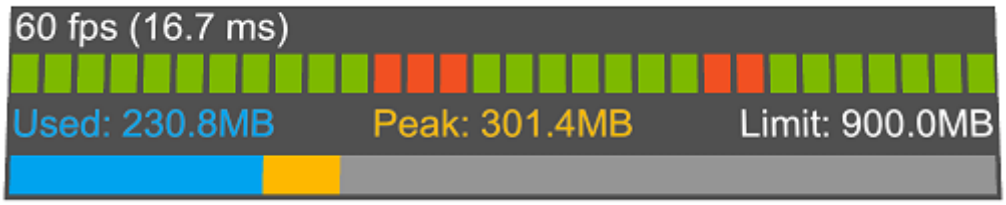


Image 7: The MRTK Visual profiler interface of Diagnostics system [27]

One of the greatest features of MRTK is that it provides an interface to test our HoloLens application in the Unity Editor. When MRTK is imported to Unity Editor, the play mode of Unity becomes a simulation of the HoloLens experience. The in-editor input simulation allows us to test virtual object behavior given a specific type of input such as hands or voice commands. This simulation takes input from keyboard. This feature accelerated the development of this project greatly for specific cases where the installation was not required (for instance checking UI elements) because it gives the developer the ability to quickly test his project without having to build and deploy it every time. However, we should not omit the build and deployment process, as this should be the final verification for the correct behavior of every aspect of our application.

Another significant feature of this toolkit is the ability to develop cross-platform applications for a variety of platforms and devices. Because the toolkit is still in its early stages, it is not easy to deploy stable apps. Therefore, it is important to understand that the developer community is small and that the toolkit's creators rely on developers' feedback. This can be very disappointing at times, but it also indicates that a new era for mixed reality applications is on the way.

The devices the toolkit supports are [1]:

- Microsoft HoloLens
- Microsoft Immersive headsets (IHMD)
- Steam VR (HTC Vive / Oculus Rift)
- OpenXR platforms

As previously stated, we developed and deployed our app for the HoloLens 1 device. Although the toolkit was not designed to be used with this device because the HoloLens 2 has been released, and this new device is the primary reason for the creation of this project, the 1 series is currently supported.

3.3 Visual Studio

We also used Microsoft Visual Studio for development. The Unity editor installation package includes a copy of Visual Studio by default. Compilers and tools for developing applications in Unity are downloaded and set automatically. The Unity Editor provide many platform configurations and switching between them is simple and straightforward. The editor then automatically configures the Visual Studio project for us to begin developing the applications.

Of course, the developer can use the Visual Studio editor to add even more settings to the project. Package handling and addition, tests, and debug operations are all enabled and ready to be used during the app deployment process.

The Nuget package manager is also used by Visual Studio to download and set up required packages and libraries. Unity has its own mechanisms for acquiring the necessary packages. Unity also has its own asset store. However, the Nuget manager is notified of all changes that occur within Unity because the Unity Editor prepares the project during build time so that it can be loaded into Visual Studio.

The Unity build procedure prepares the solution required by Visual Studio. Following that, Visual Studio compiles the solution file with the ILL2CPP compiler and builds it to generate all of the required .dll files. After completing this procedure, the application is ready to be deployed to the device.

Debugging tools as well as performance and profiling tools for monitoring the application while it is running on the device are also available for use.

3.4 .NET Framework 4.5 - 4.6 in C#

Unity and Visual Studio automatically download and install the.NET framework, which is required to develop HoloLens applications. Because app builds are extremely vulnerable to system updates, it is not always easy to determine the correct version of

the tools. These updates are required, but they may disrupt development for older devices such as the HoloLens.

The ability of applications to run on different systems and platforms is an important feature of the .NET framework. Furthermore, applications intended for publication in the Windows Store must be built with specific versions of the framework.

3.5 NVIDIA Jetson TX2

3.5.1 Jetson and its Development Kit

In our project we use NVIDIA Jetson TX2 board [29] to deploy our object detection models and establish our REST API to detect objects within a camera frame from HoloLens as well as to compare HoloLens camera frames for similarity.

Jetson TX2 is one of the fastest, most power-efficient embedded AI computing devices. This 7.5-watt supercomputer on a module brings true AI computing at the edge. It's built around an NVIDIA Pascal-family GPU and loaded with 8GB of memory and 59.7GB/s of memory bandwidth. It features a variety of standard hardware interfaces that make it easy to integrate it into a wide range of products and form factors. These characteristics make Jetson, along with the Jetson Development board an excellent choice to use as the central computer where our development server for object detection runs on, as it assures exceptional speed, power-efficiency, fast and reliable responses. The Jetson is responsible for building our object detection model and start our server in order to submit requests from HoloLens.

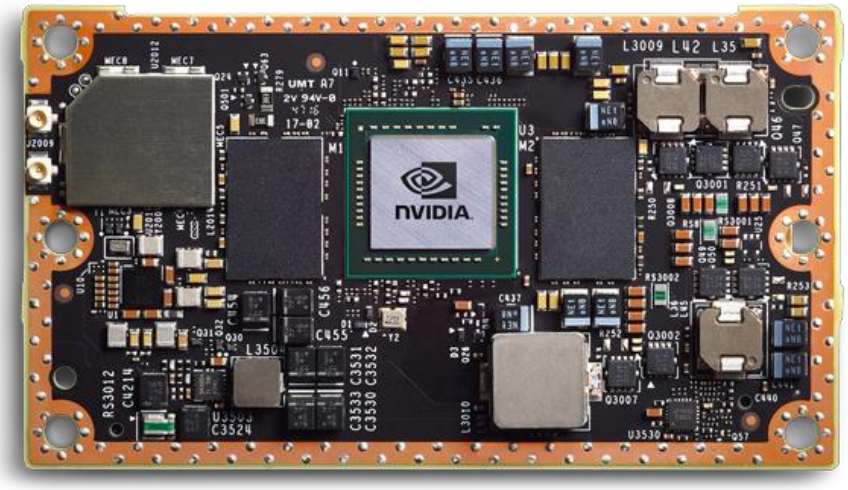


Image 8: NVIDIA Jetson TX2 Module [29]

3.5.2 Jetpack (Linux4Tegra-L4T)

Jetpack [30] is a suite of useful tools for building AI applications on top of Jetson TX2. It includes a Linux OS image for Jetson products, along with libraries and APIs, samples, developer tools, and documentation. Some tools are used directly on a Jetson system, and others run on a Linux host computer connected to a Jetson system. JetPack libraries and APIs include:

- TensorRT and cuDNN for high-performance deep learning applications
- CUDA for GPU accelerated applications across multiple domains
- NVIDIA Container Runtime for containerized GPU accelerated applications
- VisionWorks, OpenCV, and VPI (Developer Preview) for visual computing applications
- Sample applications and other libraries for visual computing tasks.

For our project we used Jetpack version 4.5 that comes with a customized Linux distro named Linux4Tegra (L4T) 32.5.0.

3.5.3 NVIDIA Docker container for Machine Learning purposes

NVIDIA has provided us many wonderful resources, including docker images for Machine Learning purposes. Docker [33] is an open-source software platform to create, deploy and manage virtualized application containers on a common operating system (OS), with an ecosystem of allied tools. All tools needed for implementing our object detection system functionalities can be delivered through prebuilt Docker images containing L4T, and ML libraries. The use of Docker constitutes an alternative solution to native installments on Linux4Tegra (L4T) OS (a modified version of Ubuntu 18.04). Docker is a very convenient tool, as whenever we need to reproduce our project, it can help us to get rid of all installation, setting up, and dependencies.

For our needs, we downloaded and installed NVIDIA L4T ML docker container [32], that can be found NVIDIA GPU CLOUD14 (NGC) [31], a repository containing Docker Images. The L4T ML docker image contains TensorFlow, PyTorch, JupyterLab, and other popular ML and data science frameworks such as scikit-learn, scipy, Pandas, numpy and OpenCV pre-installed in a Python 3.6 environment. Detailed information regarding the scripts used to start this container with specific configurations related to our development server can be found in a following chapter.

3.6 Python for object detection scripts

We used Python to write our scripts running on NVIDIA Jetson TX2 used to create a REST API for applying object detection on a camera frame from HoloLens device to get the detected objects and obtain a similarity score between two camera frames. We choose the Python programming language because of its simplicity. The development server for our API calls is developed with the help of Python library Flask.

Flask [34] is a small and lightweight Python web framework that provides useful tools and features that make creating web applications in Python easier. It gives developers flexibility and is a more accessible framework for new developers since we can build a web application quickly using only a single Python file. Flask is also extensible and doesn't force a particular directory structure or require complicated boilerplate code before getting started. When object detection functions are applied in a HoloLens camera frame, the results are returned back to the HoloLens in JSON format.

Object detection and frames similarity script functions are developed with the help of OpenCV. OpenCV [35] is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as Numpy which is a highly optimized library for numerical operations, then the number of weapons increases in a developer's hands.

4. MIXED REALITY FEATURES

The fundamental concepts required to understand how our application, or any MR application works are presented here. The HoloLens receives user input in the form of gestures, gaze, and voice commands. In an application, user input can be used to manipulate holograms. Gestures and input in general differ between the first and second generation of the HoloLens. This section presents the main gestures for both generations.

4.1 Holograms

HoloLens lets you create holograms, which are objects made of light and sound that appear in the world around you like real objects. Holograms respond to your gaze, gestures, and voice commands. They can even interact with real-world surfaces around you. With holograms, you can create digital objects that are part of your world. The holograms that HoloLens renders appear in the holographic frame directly in front of the user's eyes. Holograms add light to your world, which means that you see both the light from the display and the light from your surroundings. When you have a particular location for a hologram, you can place it precisely at that point in the world. As you walk around, the hologram appears stable based on the world around you. If you use a spatial anchor to pin the object, the system can even remember where you left it when you come back later. Some holograms follow the user instead, positioning themselves based on the user no matter where they walk. You may even choose to bring a hologram with you for a while and then place it on the wall once you get to another room. Holograms can also be occluded by real-world objects. For example, a holographic character might walk through a door and behind a wall, out of your sight.

Since HoloLens is a Mixed Reality device, holograms blend with your physical environment to appear and sound like they're a part of your world. Even when surrounded by holograms, you can see your surroundings, move freely, and interact with other people and objects.

But holograms aren't only about light and sound, they're also an active part of your world. Gaze at a hologram and gesture with your hand, and a hologram can start to follow you. Give a voice command, and the hologram can reply. As HoloLens knows where it is in the world, a holographic character can look at you directly in the eyes and start a conversation with you. A hologram can also interact with your surroundings. For

example, you can place a holographic bouncing ball above a table. Then, with an air tap, watch the ball bounce, and make sound as it hits the table.

Image 9: An example of working with Holograms [36]

The sensors in the HoloLens can see a few feet to either side of you. When you use your hands, you must keep them within that frame or HoloLens will not see them. This frame moves with you as you move. However, because mixed reality is still a emerging technology, HoloLens has some limitations. Hand position or object placement, for example, is not always correctly recognized. This should be considered when developing for the Microsoft headset to ensure that the difference in inches does not have a significant impact on the app's functionality.

4.2 Holograms and user comfort

There are steps that human eyes have to take for them to see objects clearly. Two of them are important when it comes to AR and VR headsets. The first one, accommodation, is when an eye adjusts it’s focus on a distance to see a clear image. The second one, vergence, is the eyes’ focus relative to each other. The eyes must

converge to an object's distance not to see double images. Naturally eye accommodation and vergence are linked, but with HMDs, the user's eyes will accommodate to the distance of the display while the vergence will change according to the distance of the object of interest. This breaks the link between accommodation and vergence and may cause eye strain and discomfort.

Even though the visual experience delivered by HoloLens is obviously amazing, it must be delivered properly in order to avoid a frequently encountered phenomenon, known as cybersickness. This illness, also called virtual reality sickness, may occur while a user is inside the VR system and experiences eye discomfort, headache, nausea, dizziness, and disorientation.

Here is where a near clip plane becomes helpful. When using Unity, it is extremely important to set the near clipping plane to no less than 0.85m. If it is less than this, then objects will appear far too close to the user's eyes and cause them to cross. In addition to making the visual experience difficult to deal with, it can cause severe discomfort for the viewer. Also, the ideal distance from holographic content to the viewer's eyes should be kept as close to 2m as possible because this distance is free from a binocular rivalry.

Two pictures projected onto the left and right part of the transparent combiner lenses will be fully overlapped at 2 meters from the user, therefore this distance is considered as the optimal one. In cases when it is impossible to place holographic content at this recommended distance, you can use the most favorable zone, which extends at a range from 1.25 to 5 meters away from the user.

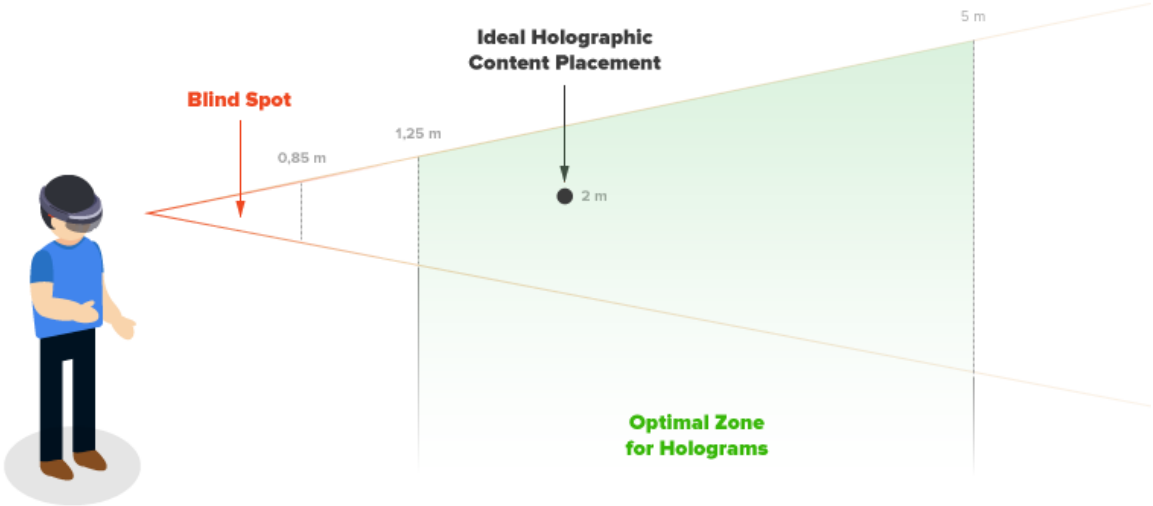


Figure 9: HoloLens device field of view in the real world [37]

Keeping the application's FPS at a steady 60 is crucial for a smooth and comfortable experience when there are moving objects in a scene. As an application's framerate drops, judder appears in moving objects in the form of uneven motion and double images. In addition to high FPS, framerate consistency is important as well. Consistent FPS of 30 looks more stable than FPS fluctuating from 60 to 30 and back [38].

4.3 Spatial coordinate systems

At their core, mixed reality apps place holograms in our world that look like and sound real objects. This involves precisely positioning and orienting those holograms at meaningful places in the world, whether the world is their physical room or a virtual realm you've created. Windows provides various real-world coordinate systems for expressing geometry, which is known as spatial coordinate systems. We can use spatial coordinate systems to reason about hologram position, orientation, gaze ray, or hand positions [39].

HoloLens uses Cartesian coordinate system (X, Y, Z) with metric values. Spatial coordinate systems express their coordinate values in meters. This means that objects placed two units apart in either the X, Y, or Z axis will appear 2 meters apart from one another when rendered in mixed reality. Knowing this, you can easily render objects and environments at real-world scale.

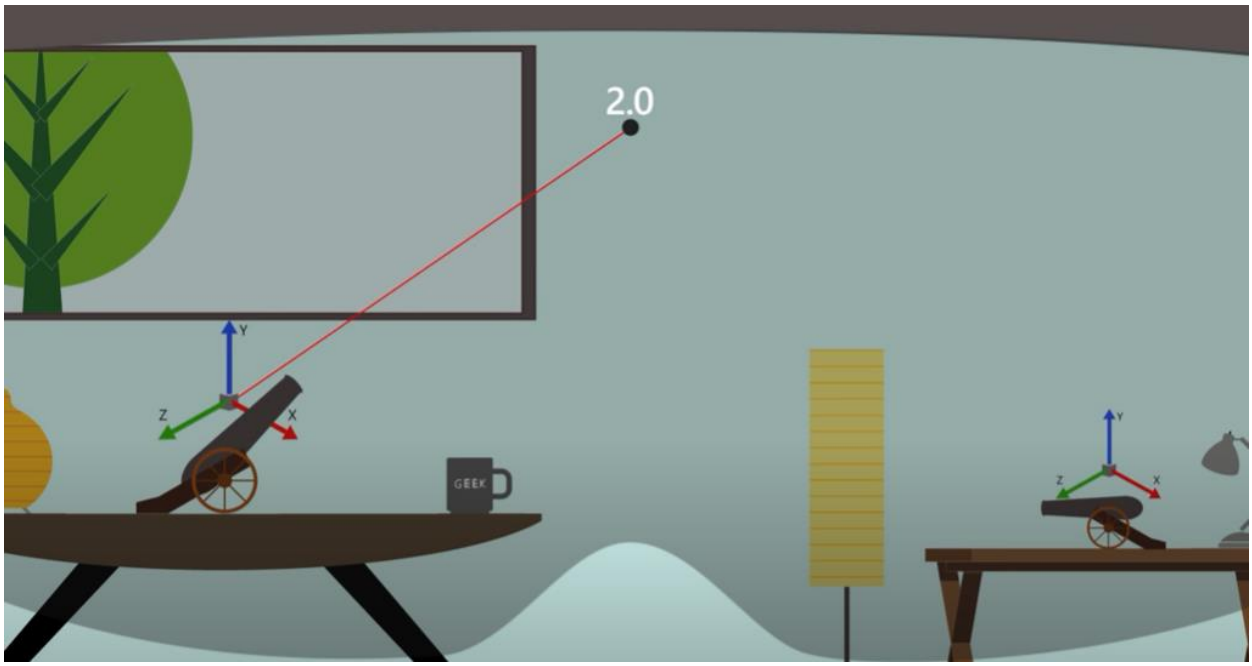


Figure 10: Spatial coordinates system [39]

The digital content that is displayed to the user can be world-locked (where the rendered object stays put when the user moves), or body-locked (where it follows the user). It is advised that most content is displayed as world-locked by default, but in scenarios where HoloLens' sensors cannot gather enough data about its environment, it is good practice to show body-locked content that would guide the user to check if something is blocking the sensors or to turn on lights, for example.

4.4 Input

As the reality of the participant adjusts more towards a MR experience through the inclusion of holograms, the aspect of interaction becomes essential. Towards that end, HoloLens comes integrated with multiple capabilities for interacting with the holograms as well as understanding the surrounding physical environment. By exploring the aforementioned capabilities, several key forms of input emerge.

4.4.1 Gaze

Perhaps the most noteworthy form of input is the way HoloLens understands the position of the participant in the world as well as the direction they might be looking at, also known as gaze. In mixed reality appliances, the fundamental means for of both targeting and user input is gaze [40]. By taking advantage of the position and orientation data of participant's head, HoloLens is capable of determining the head gaze vector. This can be described as a sort of ray pointing straight forward from directly between the participant's eyes. Furthermore, the ray can be utilized in holographic applications developed for HoloLens to pinpoint holograms as well as to determine whether the participant is looking at a virtual or real-world object.

To stimulate participant's intention in an application created for HoloLens, a very common practice is to indicate the gaze with continuous visual feedback such as a cursor. A cursor is a visual indicator of the participant's current targeting vector, providing continuous feedback to aid the participant in understanding where their focus is at all times, as well as to indicate what possible hologram, area, or other point of interest might respond to input, as shown in Image 10. It is possible to further utilize the cursor to target an object inside an application and attempt at interacting with it.

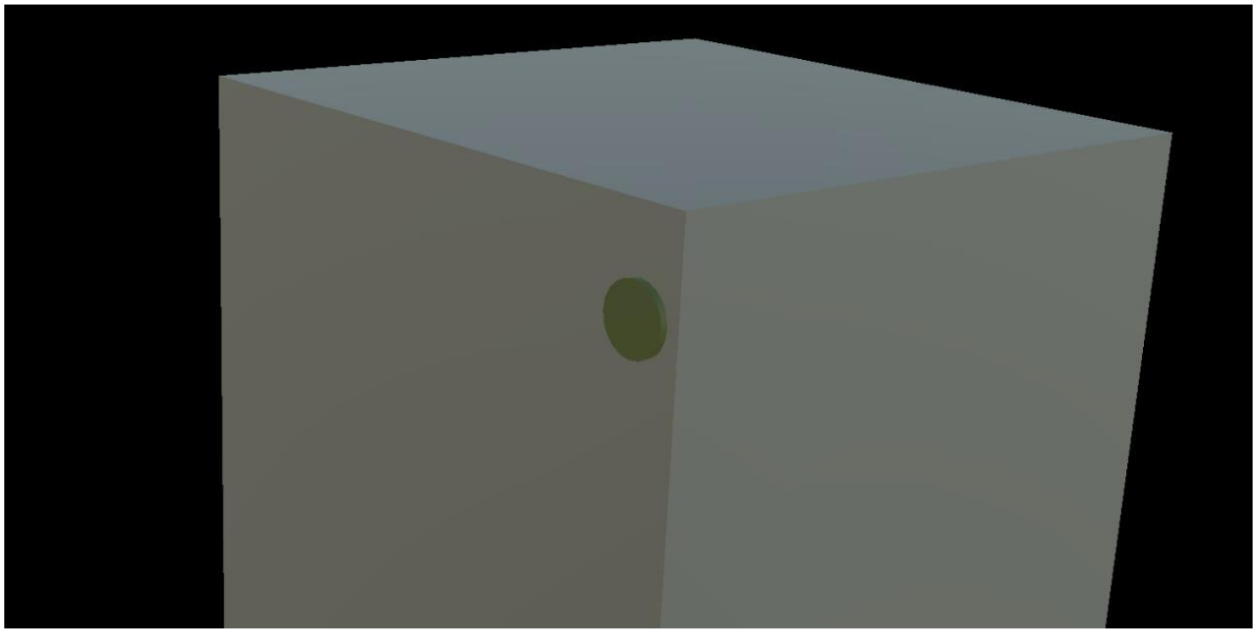


Image 10: Depiction of a primitive cursor, a green tiny circle aligned on top of a cubic object in 3D space, visually indicating gaze

Gestures and voice, as well as motion controllers are considered the essential means for the participant to perform interactions in mixed reality.

4.4.2 Gestures

Whereas gaze is mostly responsible for determining the point of interest in a holographic application, gestures complete the formulae for interaction by introducing the element of acting. Interaction is formed by utilizing participant's gaze to target and hand gesture or voice to act upon a targeted hologram [41]. The types of gesture sources recognized by HoloLens headsets include hand and voice.

Out of the two gesture types recognized by HoloLens, the more conventional method for interaction could be considered the hand gestures, which can be further subdivided into two core component gestures called air tap and bloom. Air tap is a type of gesture in which participant taps their index finger with their hand held upright, as seen in Figure 9. It is a universal action designed to perform a select type of action, similarly to conventional mouse clicks used in interacting with traditional computer applications.

Bloom is a special system action designed exclusively for HoloLens. The sole purpose for it is to bring up the holographic start menu (a central system menu containing a list or category of settings and applications installed on the system). The functionality of the bloom gesture can be considered identical to pressing the Windows key on a keyboard plugged into a system running on a Windows operating system.

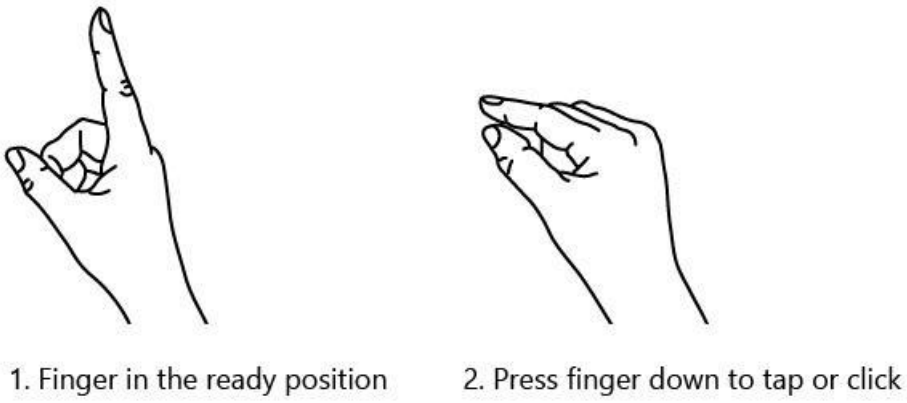


Figure 11: Performing air tap hand gesture [42]

In order to execute a bloom gesture, the participant is instructed to hold out their hand with the palm facing upwards whilst keeping their fingertips enclosed, and then open the hand in a releasing fashion. The procedure is depicted in Figure 11.

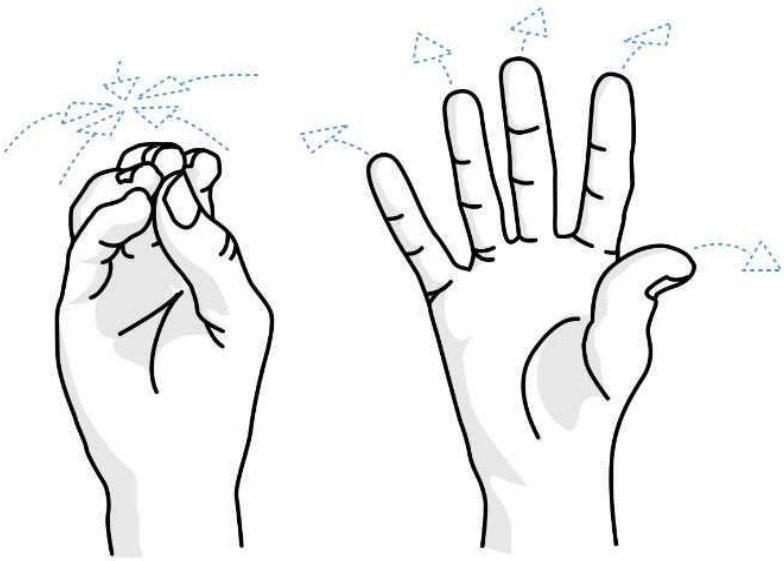


Figure 12: Performing bloom hand gesture [42]

In addition to the two core hand gesture components, HoloLens provides support for several more advanced composite hand gestures.

The first gesture is hold, which is quite similar to an air tap in that the gesture starts from the index finger of the participant being in ready position, and then proceeds to perform a continuous tap by holding the finger pressed down. Whilst the participant is holding down the index finger, the gesture can be utilized to perform a range of actions such as

initiating relocation of a hologram or pausing a hologram. This combination gesture can be considered equivalent to holding down a mouse button.

Figure 13: Hold Gesture [42]

The secondary gesture is manipulation, which can be considered an extended version of the hold gesture. Similar to the hold gesture, manipulation begins by participant holding down their index finger. The gesture can then be continued by participant moving their hand freely whilst keeping the finger held down, allowing the targeted hologram to react one by one to the participant's hand movements. The manipulation gesture can be used for performing actions including relocation, resizing, or rotation of a hologram.

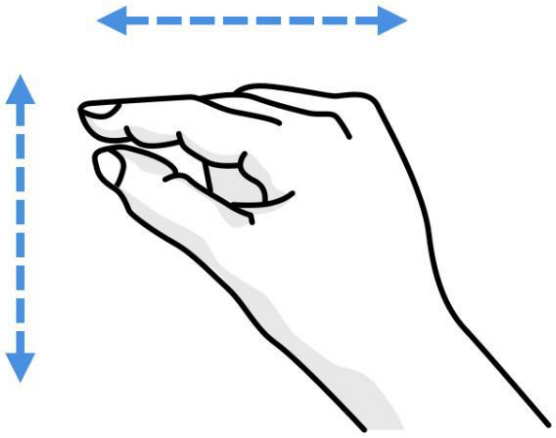


Figure 14: Manipulation Event [42]

The final, and perhaps the most complex composite hand gesture is the navigation gesture. The functionality of the navigation could be described equal to operating a virtual joystick. Navigation begins similarly to a hold gesture by the participant tapping and holding down the index finger, and then proceeding to move their hand along three separate axes (such as X, Y, and Z), ranging between -1 to 1, with 0 being the starting

point, therefore creating a normalized 3D cube centered around the initial press. It is possible to utilize navigation to create constant velocity-based scrolling or zooming gestures, equivalent to an example of using a mouse to scroll a 2D UI by pressing down the middle mouse and then shifting the mouse back and forth.

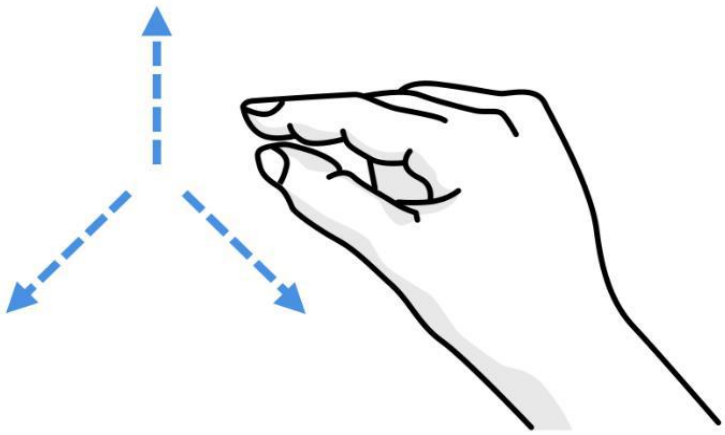


Figure 15: Navigation Event [42]

Performing hand gestures takes place within a gesture frame. The frame is an area generated in front of HoloLens by the gesture-sensing cameras and de-fines the boundaries in which the device can detect the gestures performed by the participant. The frame is set roughly from nose to waist depending on the height of the participant, as illustrated in Figure 16.

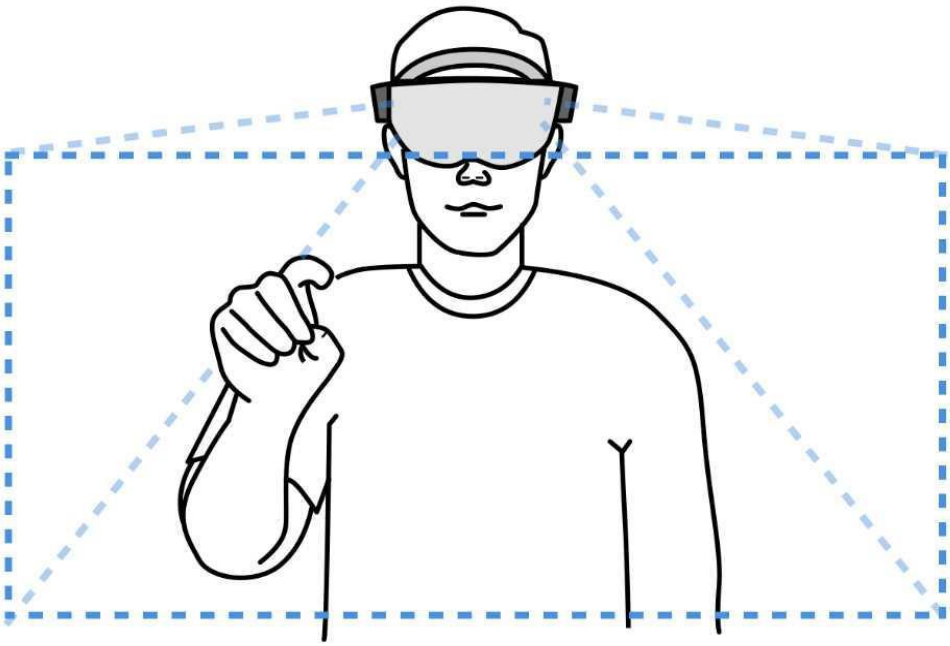


Figure 16: Gesture frame [42]

Should a gesture such as manipulation or navigation take place completely or partly outside the gesture frame, as soon as the gesture cannot be detected, the HoloLens will lose the input [43]. Recognition of the hand gestures performed within the gesture frame can occur when either or both hands of the participant are visible to HoloLens, as in the hands of the participant being in line of sight of the gesture-sensing cameras. Furthermore, the recognition is affected by orientation of the participant's hands, as HoloLens can see them when they are either in ready state (hand reached out with back facing to-wards the participant and index finger up) or the pressed state (hand reached out with the back facing the participant and the index finger down). Should the hands of the participant be in any other pose, HoloLens will lose track of them.

4.4.3 Voice

In addition to the two key forms of input of gaze and gestures, HoloLens supports voice as source for receiving input from the participant. Voice enables the participant to directly command a hologram without the need for utilizing separate hand gestures, allowing in many cases for a much quicker and simpler way to interact with the targeted hologram. Voice can be considered excellent at managing complicated interfaces, as the participant may use a single voice command to quickly cut through the nested menus [44].

HoloLens comes equipped with multiple built-in commands universally recognized by the platform. Perhaps the most basic of the commands is the select command. Select behaves equivalently to an air tap gesture, allowing the user to activate holograms or other holographic elements (such as UI elements) with a single line of speech. HoloLens then confirms the voice commands by letting the participant know that the command has been executed by displaying a tooltip with "Select", as well as cueing a sound effect. Additional HoloLens-specific voice commands include but are not limited to examples such as "What can I say?", "Go home", "Launch", "Move", and "Take a picture".

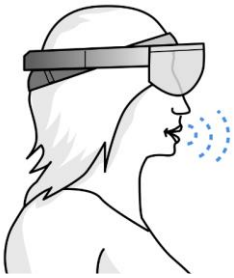


Figure 17: Voice Commands and the HoloLens Platform [44]

As can be observed from the aforementioned examples, the voice commands are rather self-explanatory in describing their functionalities. Finally, HoloLens enhances the experience of using voice commands by dis-playing labels on various UI elements, telling the participant what voice commands they can use to interact with the elements. By using gaze to target and highlight a button in a holographic application (such as the “Exit” button frequently located in the top right corner of an application), the participant will be notified if the button possesses a voice command attached to it, by displaying a label above it, as shown in Image 11. This model of voice input is known as “see it, say it”.

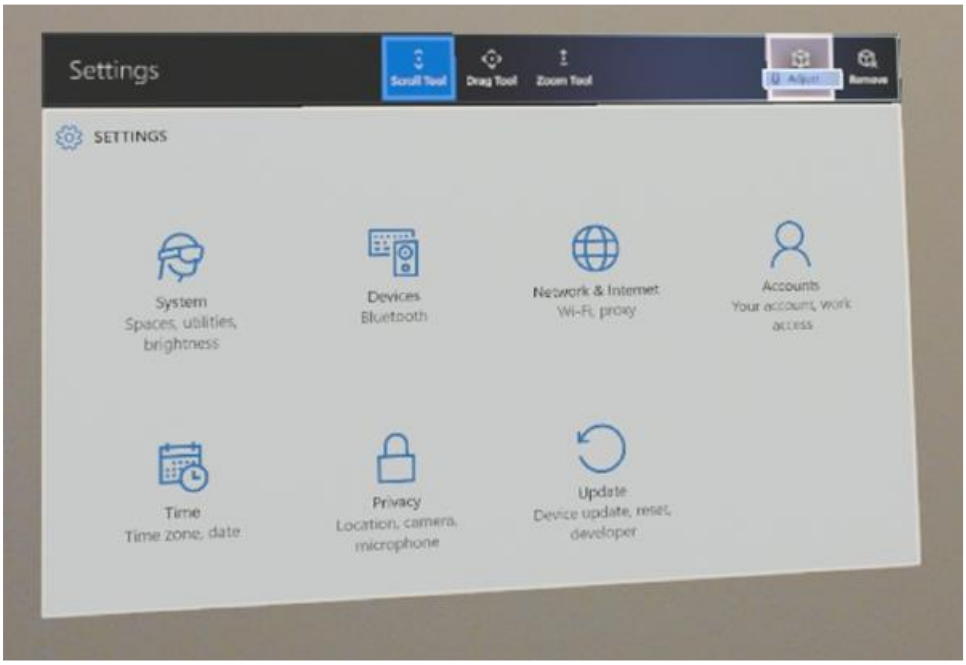


Image 11: "See it, say it" label in the top right corner of a holographic application [44]

This sort of approach is highly recommended to follow when developing applications for HoloLens, as the participant can easily understand what to say to control the system more efficiently, and therefore receive a better experience.

For our application we have created a listener for the user’s tap gesture to initiate the environment scan process and detect objects. For the app’s main menu buttons new custom voice commands have been introduced for their selection and a global voice command “Go to menu” has also been registered in order to return to the main menu from any internal scene. Finally, to visualize the user’s gaze, in Visual and Audio scan scenes of the app (described in detail in a following chapter), we also have a cursor which is green when the user can initiate a new scan and red while a scan is in process and the results receival is pending.

4.5 Spatial Mapping and Anchors

Similar to understanding physical gestures performed by the participant, HoloLens is capable of understanding its surrounding physical environment. This feature is known as spatial mapping. Four environmental cameras on the front of the HoloLens are utilized to map the physical surroundings and objects to generate a 3D model of the real-world. This feature allows HoloLens to stand out from rest of the AR devices as a MR device [45].

Spatial mapping feature consists of two primary types of components, a spatial surface observer and a spatial surface. When utilized in a holographic application, the spatial surface observer can be described as the fundamental contributor of spatial mapping, tasked with the procedure of scanning of the physical surfaces. For each scanned real-world surface, the application utilizes what's known as spatial surfaces, a tiny volume of space represented as triangular meshes combined together to generate a complete digital mesh of the surroundings as seen in Image 12.

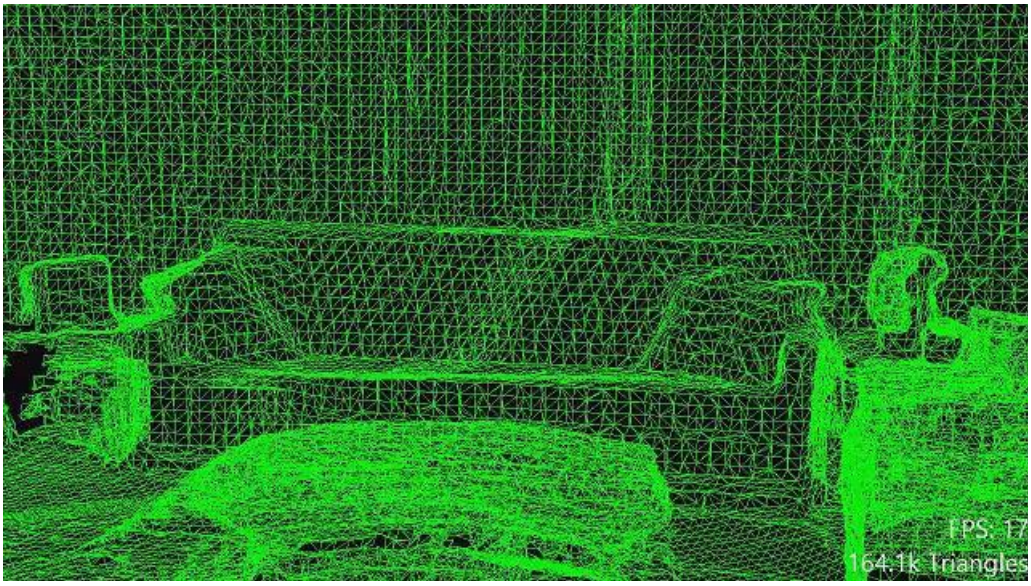


Image 12: Model of a room, generated by spatial mapping [45]

Whilst using HoloLens, the surrounding physical environment of the participant is continually scanned, and from the scan results HoloLens generates a 3D model matching that of the space the participant is currently inhabiting. The model is actively updated as spatial surface observer studies the environment for changes, and if necessary, augments the model by adding new spatial surfaces from the physical real-world counterparts captured by the environmental cameras and removing spatial surfaces that no longer exist within the view bounds of the spatial surface observer.

Additionally, the developer edition of HoloLens comes with a tool known as “Device Portal” (a web browser-based application containing various settings and features to remotely monitor and interact with the HoloLens), which allows the participant to view the generation of the model in real time, as can be seen in Image 13.

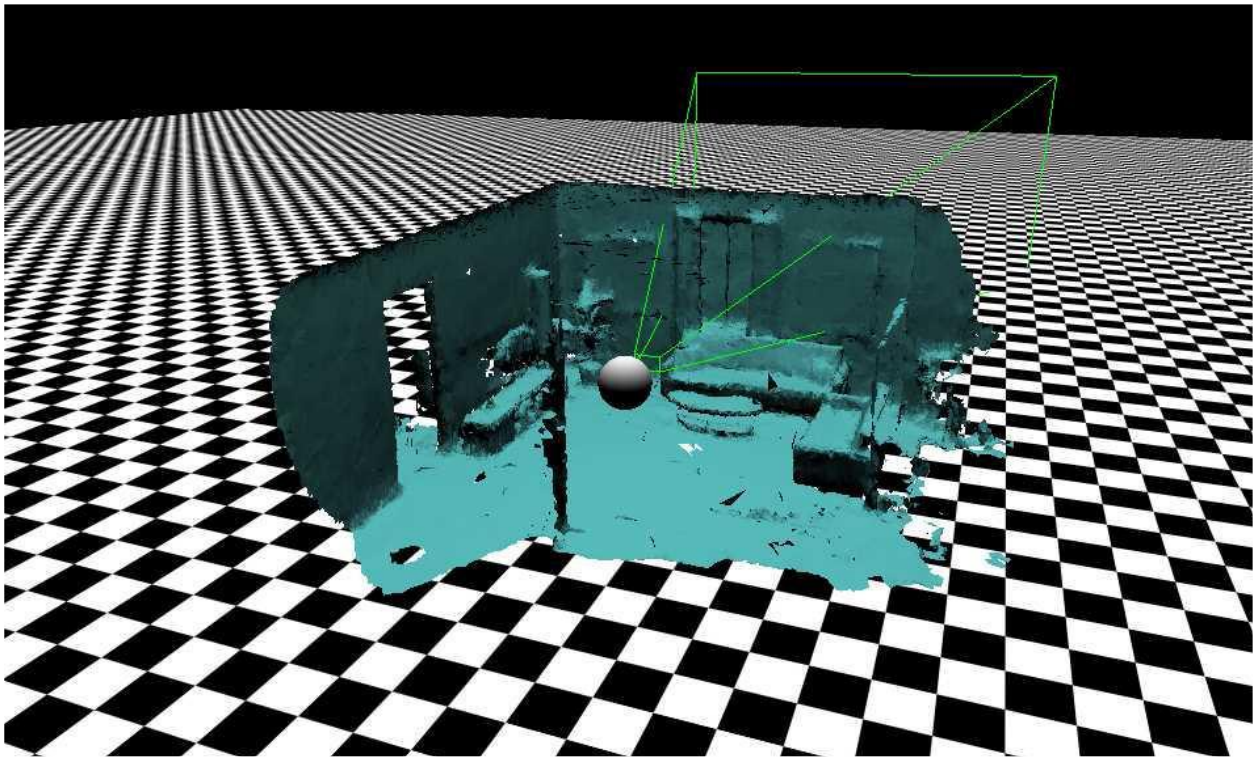


Image 13: Real time feed of spatial mapping in progress [43]

Furthermore, for each holographic application, the spatial surface observer is required to define one or more bounding volumes in which the spatial mapping may occur. In programmatic sense, the shape of bounding volume may also vary, and the shape can be modified to match that of a box or a sphere, for example. In terms of positioning, the volumes can be configured to be fixed (a static position with respect to the real-world) or they can be hooked up to the HoloLens, so that they move in tandem with the HoloLens as the participant moves about in the physical environment.

For the holograms to behave naturally with the physical environment and create a convincing illusion, such as having a digital ball roll smoothly on a physical floor, it is necessary for a holographic application to be aware of both virtual and physical realities. A very accurate positioning of the holograms is of paramount importance, so that the immersion of the participant remains coherent. For this purpose, HoloLens utilizes spatial coordinate system to calculate the positional interactions between holograms and the physical surfaces.

In order to understand the complexity of spatial coordinate systems for HoloLens, one is urged to consider a similar system utilized in a completely virtual environment. In a virtual application, a single master-coordinate system can calculate the positioning of every object part of the experience very precisely, as the elements of physical world are out of scope. Every object part of the application is able to map and relate to same coordinate system. This can lead to a very stable and precise experience for the participant. However, in case of HoloLens and holographic applications, in which physical objects are in scope of the experience due to spatial mapping, the coordinate system must be able to calculate the positioning of holograms located in a physical environment. This can prove a synchronization challenge, as the spatial mapping utilized to digitalize the physical environment might not scan the surroundings accurately, leading to either incomplete or inaccurate virtual model of the space, which may further affect the position between holograms, as well as cause other malfunction (such as making the holograms float in air or suddenly shift positions). Consequently, the participant may become motion-sick at worst.

However, HoloLens comes with a feature capable of countering the issue, known as spatial anchors. To illustrate a pivotal position in the world of which the system should be aware of over time, a spatial anchor may be utilized. The spatial anchors receive individual coordinates within the system that are managed if necessary to ensure the anchored elements remain still in their place [46]. Spatial anchors may prevent drifting of holograms as well as ensure that they remain at their designated positions, even as the spatial mapping updates the model of the space.

Other meaningful features and uses of spatial mapping include occlusion and visualization. Perhaps the more important is the occlusion, which is utilized to occlude holograms. In order to improve the experience of reality in a holographic application, spatial mapping can occlude holograms by either hiding them completely or partially from the view of the participant. For example, a hologram positioned behind a physical wall should be considered invisible to the participant in order to increase perceived reality. This, of course, presumes that the wall has been scanned and included in the spatial map model, as the hologram would be visible through the wall otherwise. However, sometimes, it can be considered undesirable to occlude holograms. Should the participant wish to interact with a hologram, it would have to be visible in some way. One method achieving this could be by rendering the hologram diver-gently when it is occluded by the spatial mapping (such as altering the level of brightness). In that case,

the hologram would be visible enough for the participant to visually locate it, whilst still being obscured enough to provide the im-pression of being hidden behind something.

Being a MR experience in which physical real-world objects provide fundamental base for holograms to interact with, most of the times it can be considered appropriate for spatial surfaces generated by spatial mapping to be invisible, as in to reduce visual clutter and allow the real-world speak for itself. However, sometimes it can prove meaningful to visualize spatial surfaces regardless of the real-world counterparts being already visible [45].

This could prove especially useful in cases in which the hologram and the participant seek to share understanding of something (such as determining whether a physical surface is solid or not). An example case could be a hologram of a painting. When attempting to place on a wall, it should provide visual feedback whether it is able to settle on the wall or not. It is possible to determine this by rendering the individual spatial surfaces behind the hologram differently, thus producing a “ground” effect by emitting a shadow onto the surface [45]. This would allow the participant to have a much sharper feel of the precise physical distance be-tween the hologram and the surface. A more general example of a similar practice could be depicted by participant visually previewing a change before committing to it (such as previewing a picture before printing it).

The use of physics simulation is another way in which spatial mapping can be used to reinforce the presence of holograms in the user's physical space. Physics simulation also provides the opportunity for an application to use natural and familiar physics-based interactions. Moving a piece of holographic furniture around on the floor will likely be easier for the user if the furniture responds as if it were sliding across the floor with the appropriate inertia and friction.

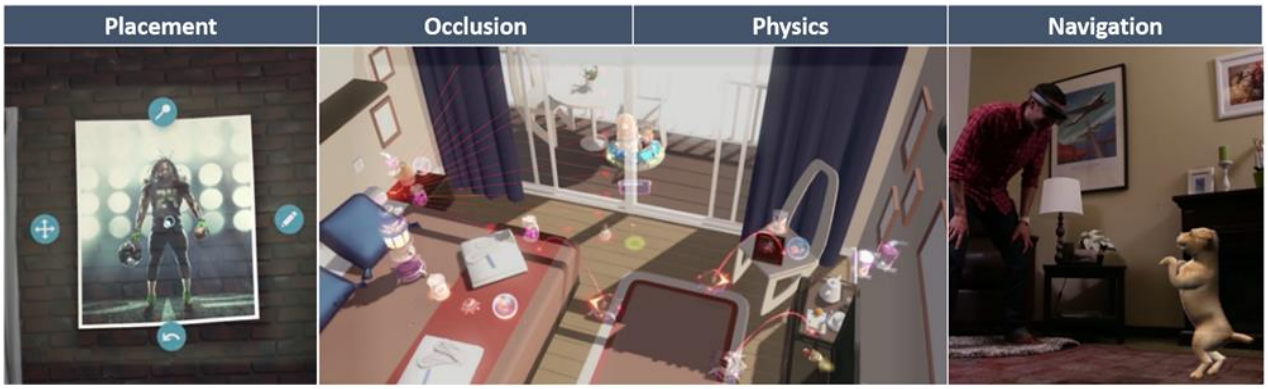


Figure 18: Common usage scenarios of spatial mapping [45]

Applications can use spatial mapping data to grant holographic characters (or agents) the ability to navigate the real world in the same way a real person would. This can help reinforce the presence of holographic characters by restricting them to the same set of natural, familiar behaviors as those of the user and their friends. Navigation capabilities could be useful to users as well. Once a navigation map has been built in a given area, it could be shared to provide holographic directions for new users unfamiliar with that location.

Spatial mapping and scene understanding are Mixed reality features of great importance for our application, as they are necessary for an essential process of placing our holograms for each object detection result. For each result, after transforming the 2D pixel coordinates in 3D world coordinates, we need to perform a raycast to the spatial mesh in order to find the collision point between this ray and the real scene and obtain the position for our holograms (object detection bounding boxes and tooltips for their class name).

4.6 Spatial Sound

When a user moves through a HoloLens experience, they can’t keep track of everything happening around them. This is partly because the human mind can only handle so many inputs at the same time, and partly because of the limited field of view on the device. Sound can provide a footing and reference for the experience that helps the user both navigate and trust it.

In a HoloLens experience, the sound simulates the environment by providing both distance and direction. The highly accurate and real-time speakers on the HoloLens delivers audio as if it was real in the experience. While the visuals are limited to where the user is looking and the physical frame of the lenses on the device, audio can come from all directions [47].

Our human ears are already brilliant at managing sounds coming from all directions and distances. This spatial sound is part of our natural universe, and to create a HoloLens experience that is both natural and convincing, it is paramount that audio functions in a similar way. This is done through spatial sound, which means sound emanates from a particular location, and as the user moves around the space, the sound stays in the same location. This is an integral part of the HoloLens experience, and while not part of the GGV paradigm, is just as important as gaze, gesture, and voice.

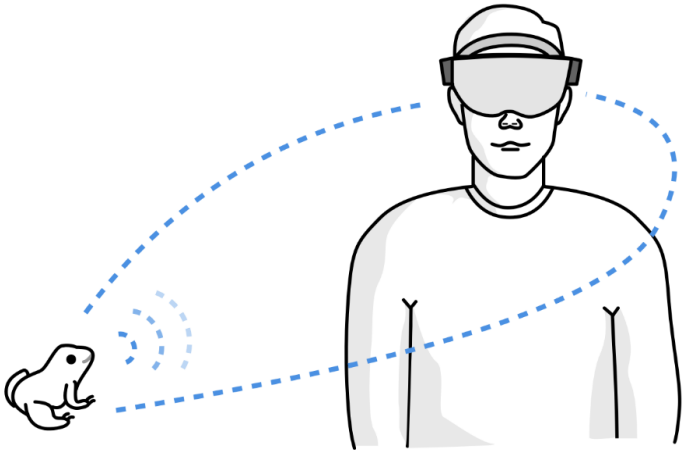


Figure 19: Spatial Sound [47]

When we use Spatial Sound in an application, this locates sounds in a 3 dimensions space all around the user that uses the HoloLens. Sounds will then appear as if it came from real physical objects or holograms of mixed reality in a user environment. Spatial Sound helps to create a much more credible and immersive experience.

In our application, we use spatial sound feature for Audio scan to announce a descriptive message of the environment scan results with the detected objects and their relative position from the user, along with their respective distances from the user. Furthermore, we use sound effects when the environment scanning process initiates, as well as when the process ends with success or failure.

4.7 Follow Toggle

We needed to add some functionality to make the UI follow the users view and specifically the main menu buttons collection. If we had left the UI without some behavior regarding this aspect, the user would lose sight of the UI if he committed any type of movement. This would happen because the anchors of the UI would not change when the user moves. We wanted to implement the behavior like the HoloLens menu, where the menu followed the user. This is important in order to make the UI visible to the user even if the user has moved his head around and enhances user experience. After some time, the UI follows the user's movement and view tracking any movement

in any direction. For example, if the user lifts his head up, after some time the UI will follow his view.

This functionality is achieved using the MRTK Follow toggle class [48]. The Follow class positions an element in front of the of the tracked target relative to its local forward axis. The element can be loosely constrained (also known as tag-along) so that it doesn't follow until the tracked target moves beyond user defined bounds. This class also provides mechanisms to configure the orientation, distance, and direction of the target.

Detailed information regarding this configuration for our application are provided in a following chapter. In image we can see this follow functionality for the main menu buttons collection from our application.

Image 14: Follow toggle for our application's main menu button collection

5. OBJECT DETECTION WITH YOLOV3

In this chapter the theoretical background of visual object detection, with emphasis on the convolutional neural network (CNN) YOLOv3 model is presented.

5.1 Visual object detection

Object detection is a computer vision technique that allows us to identify and locate objects in an image or video. With this kind of identification and localization, object detection can be used to count objects in a scene and determine and track their precise locations, all while accurately labeling them. Specifically, object detection draws bounding boxes around these detected objects, which allow us to locate where said objects are in (or how they move through) a given scene.

Object detection is commonly confused with image recognition, so before we proceed, it's important that we clarify the distinctions between them. Image recognition assigns a label to an image. A picture of a dog receives the label "dog". A picture of two dogs, still receives the label "dog". Object detection, on the other hand, draws a box around each dog and labels the box "dog". The model predicts where each object is and what label should be applied. In that way, object detection provides more information about an image than recognition. In Figure we can see an example of how this distinction looks in practice.

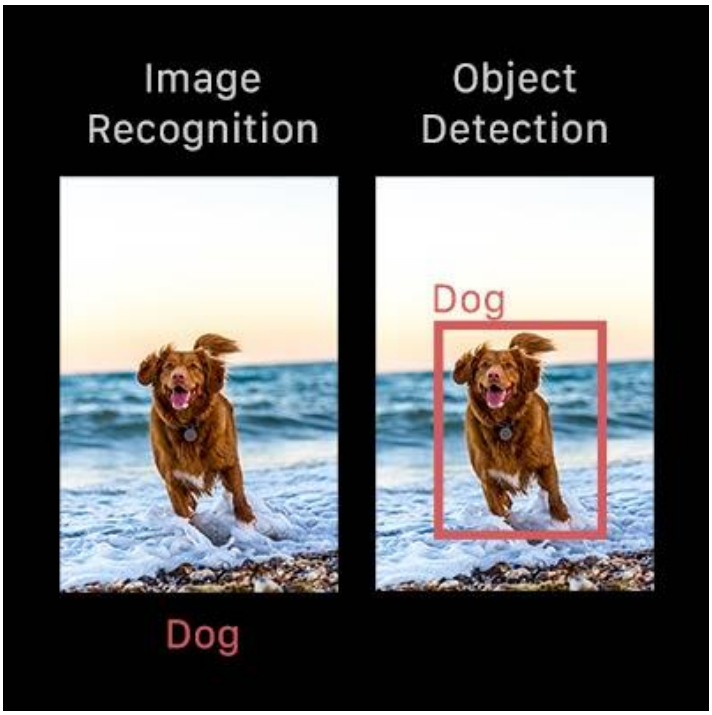


Figure 20: Visual representation of Image Recognition vs Object: Visual representation of Image Recognition vs Object Detection [49]

Object detection is inextricably linked to other similar computer vision techniques like image recognition and image segmentation, in that it helps us understand and analyze scenes in images or video. But there are important differences. Image recognition only outputs a class label for an identified object, and image segmentation creates a pixel-level understanding of a scene's elements. What separates object detection from these other tasks is its unique ability to locate objects within an image or video. This then allows us to count and then track those objects. Given these key distinctions and object detection's unique capabilities, we can see how it can be applied in a number of ways:

- Crowd counting
- Self-driving cars
- Video surveillance
- Face detection
- Anomaly detection

Of course, this isn't an exhaustive list, but it includes some of the primary ways in which object detection is shaping our future.

5.2 Modes and types of object detection

Broadly speaking, object detection can be broken down into machine learning-based approaches and deep learning-based approaches.

In more traditional ML-based approaches, computer vision techniques are used to look at various features of an image, such as the color histogram or edges, to identify groups of pixels that may belong to an object. These features are then fed into a regression model that predicts the location of the object along with its label.

On the other hand, deep learning-based approaches employ convolutional neural networks (CNNs) to perform end-to-end, unsupervised object detection, in which features don't need to be defined and extracted separately.

Because deep learning methods have become the state-of-the-art approaches to object detection, the model we chose to work with in this thesis belongs to this category.

5.3 Deep learning-based object detection

Deep learning-based object detection models typically have two parts. An encoder takes an image as input and runs it through a series of blocks and layers that learn to extract statistical features used to locate and label objects. Outputs from the encoder are then passed to a decoder, which predicts bounding boxes and labels for each object.

The simplest decoder is a pure regressor. The regressor is connected to the output of the encoder and predicts the location and size of each bounding box directly. The output of the model is the X, Y coordinate pair for the object and its extent in the image. Though simple, this type of model is limited. We need to specify the number of boxes ahead of time. If our image has two dogs, but our model was only designed to detect a single object, one will go unlabeled. However, if we knew the number of objects we need to predict in each image ahead of time, pure regressor-based models would be a good option.

An extension of the regressor approach is a region proposal network. In this decoder, the model proposes regions of an image where it believes an object might reside. The pixels belonging to these regions are then fed into a classification subnetwork to determine a label (or reject the proposal). It then runs the pixels containing those regions through a classification network. The benefit of this method is a more accurate, flexible model that can propose arbitrary numbers of regions that may contain a bounding box. The added accuracy, though, comes at the cost of computational efficiency.

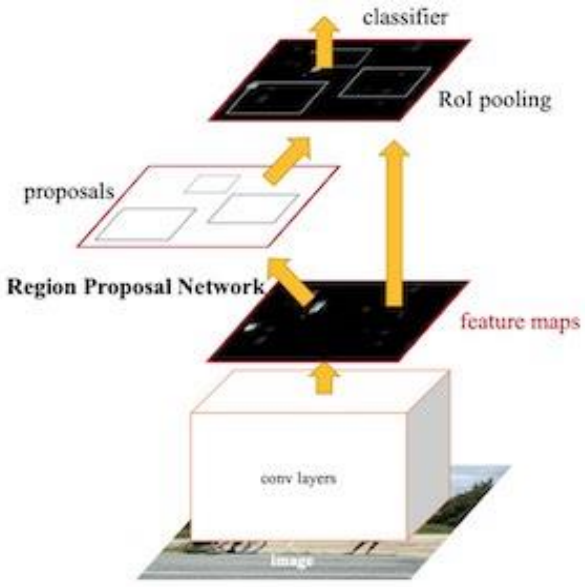


Figure 21: Representation of a region proposal network [49]

Single shot detectors (SSDs) seek a middle ground. Rather than using a subnetwork to propose regions, SSDs rely on a set of predetermined regions. A grid of anchor points is laid over the input image, and at each anchor point, boxes of multiple shapes and sizes serve as regions. For each box at each anchor point, the model outputs a prediction of whether or not an object exists within the region and modifications to the box's location and size to make it fit the object more closely. Because there are multiple boxes at each anchor point and anchor points may be close together, SSDs produce many potential detections that overlap. Post-processing must be applied to SSD outputs in order to prune away most of these predictions and pick the best one. The most popular post-processing technique is known as non-maximum suppression.

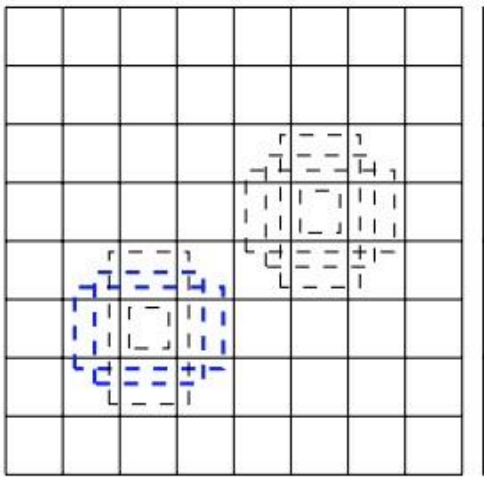


Figure 22: Single shot detectors [49]

The quality of the object detection model is commonly evaluated with mean average precision (mAP). Average precision (AP) is a standard measure for visual object detection and is widely used for detector evaluation [50]. AP can take values in the range of 0 to 1 and is defined as the area under the recall-precision curve. Recall is defined as a proportion of all true positives to all truths. Precision is a proportion of all true positives to all detections. For an object's location, the most commonly used metric is Intersection over Union (IoU). Given two bounding boxes, we compute the area of the intersection and divide by the area of the union. This value ranges from 0 (no interaction) to 1 (perfectly overlapping).

There are several models that belong to the single shot detector family. The main difference between these variants are their encoders and the specific configuration of predetermined anchors. MobileNet and SSD models [51] feature a MobileNet based encoder, SqueezeDet [52] borrows the SqueezeNet encoder, and the YOLO model

features its own convolutional architecture, which we will describe in detail in the following chapter. SSDs make great choices for models destined for mobile or embedded devices.

5.4 YOLOv3: Real-Time Object Detection Algorithm

Many object detection models take in and process the image multiple times to be able to detect all the objects present in the images. But YOLO, as the name suggests just looks at the object once. It applies a single forward pass to the whole image and predicts the bounding boxes and their class probabilities. This makes YOLO a superfast real-time object detection algorithm.

YOLOv3 (You Only Look Once, Version 3) is a real-time object detection algorithm that identifies specific objects in videos, live feeds, or images. YOLO uses features learned by a deep convolutional neural network to detect an object. Versions 1-3 of YOLO were created by Joseph Redmon and Ali Farhadi. The first version of YOLO was created in 2016, and version 3, which is used in this thesis, was made two years later in 2018. YOLOv3 is an improved version of YOLO and YOLOv2 and features multi-scale detection, a stronger feature extraction network, and a few changes in the loss function.. YOLO can be implemented using the OpenCV deep learning library.

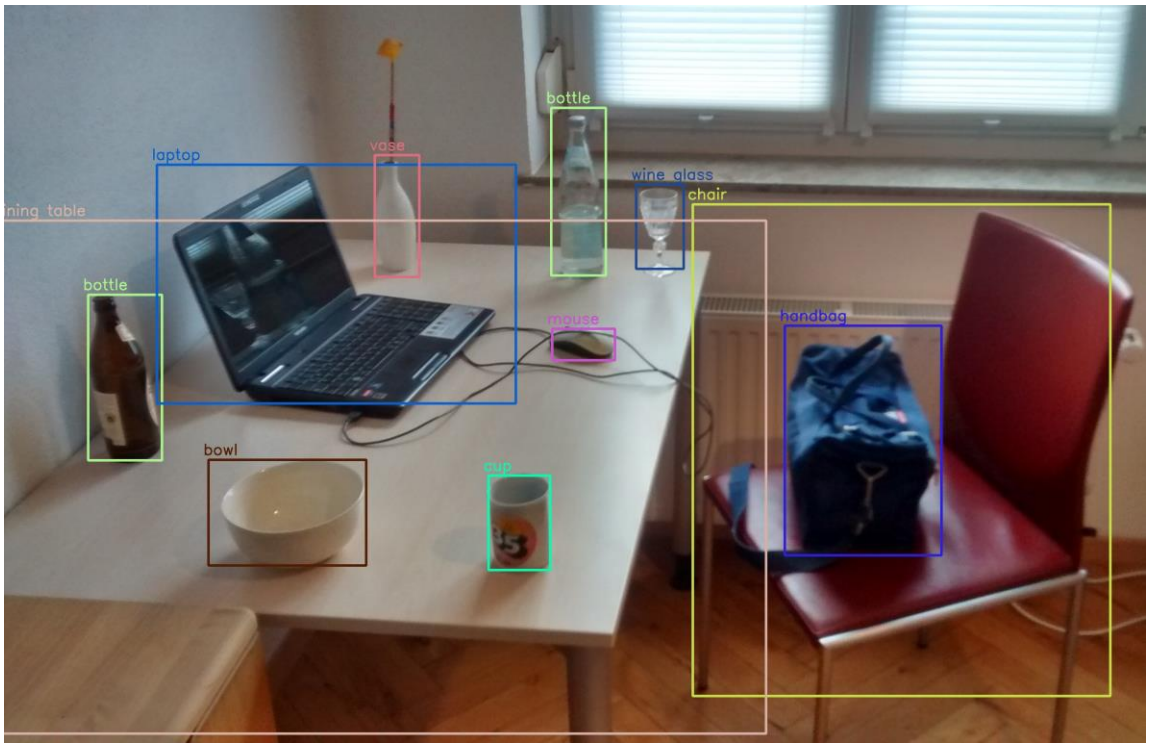


Image 15: Objects detected with OpenCV's Deep Neural Network module (dnn) by using a YOLOv3 model trained on COCO dataset capable to detect objects of 80 common classes [56]

5.4.1 Network Architecture

For understanding the network architecture on a high-level, we will divide the entire architecture into two major components: Feature Extractor and Feature Detector (Multi-scale Detector). The image is first given to the Feature extractor which extracts feature embeddings and then is passed on to the feature detector part of the network that spits out the processed image with bounding boxes around the detected classes.

The previous YOLO versions have used Darknet-19 [57] (a custom neural network architecture written in C and CUDA) as a feature extractor which was of 19 layers as the name suggests. YOLO v2 added 11 more layers to Darknet-19 making it a total 30-layer architecture. Still, the algorithm faced a challenge while detecting small objects due to downsampling the input image and losing fine-grained features.

YOLO V3 came up with a better architecture where the feature extractor used was a hybrid of YOLO v2, Darknet-53 (a network trained on the ImageNet), and Residual networks (ResNet). The network uses 53 convolution layers (hence the name Darknet-53) where the network is built with consecutive 3x3 and 1x1 convolution layers followed by a skip connection (introduced by ResNet to help the activations propagate through deeper layers without gradient diminishing).

The 53 layers of the darknet are further stacked with 53 more layers for the detection head, making YOLO v3 a total of a 106 layer fully convolutional underlying architecture. This led to a large architecture, though making it a bit slower as compared to YOLO v2 but enhancing the accuracy at the same time.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 23: The darknet-53 architecture [58]

If the aim was to perform classification as in the ImageNet, then the Average pool layer, 1000 fully connected layers, and a SoftMax activation function would be added as shown in the image, but in our case, we would like to detect the classes along with the locations, so we would be appending a detection head to the extractor. The detection head is a multi-scale detection head hence, we would need to extract features at multiple scales as well.

For visualizing how the multi-scale extractor would look like, we will take an example of a 416x416 image. A stride of a layer is defined as the ratio by which it downsamples the input, and hence the three scales in our case would be 52x52, 26x26, and 13x13 where 13x13 would be used for larger objects and 26x26 and 52x52 would be used for medium and smaller objects.

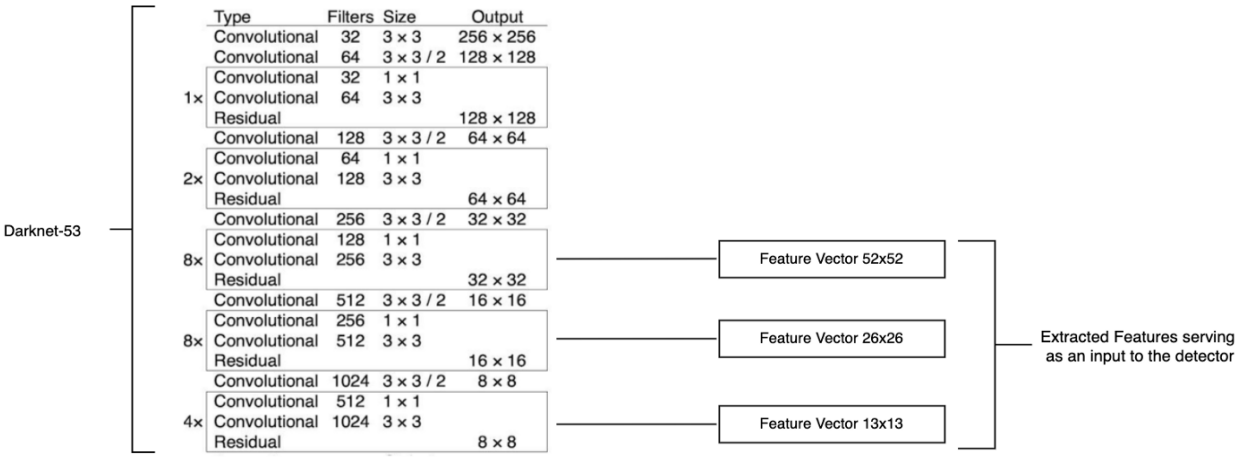


Figure 24: Multi-scale Feature Extractor for a 416x416 image [54]

An important feature of the YOLO v3 model is its multi-scale detector, which means that the detection for an eventual output of a fully convolutional network is done by applying 1x1 detection kernels on feature maps of three different sizes at three different places. The shape of the kernel is 1x1x(B*(5+C)).

As seen in Figure 25, where we take an example of a 416x416 image, the three scales where the detections are made are at the 82nd layer, 94th layer, and 106th layer.

For the first detection, the first 81 layers are downsampled such that the 81st layer has a stride of 32 (as mentioned earlier, a stride of a layer is defined as the ratio by which it downsamples the input) resulting in our first feature map of size 13x13 and the first detection is made with a 1x1 kernel, leading to our detection 3D tensor of size 13x13x255.

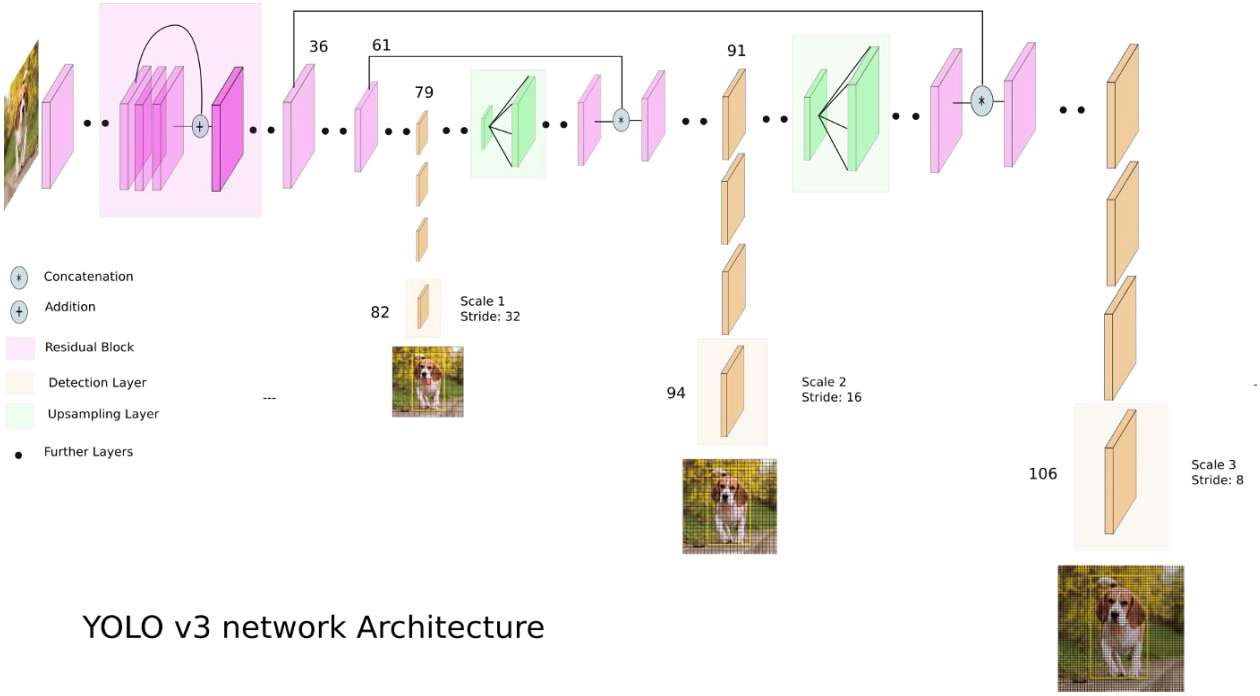


Figure 25: Complete YOLOv3 Architecture [59]

For the second detection, the 79th layer onwards is subjected to convolutional layers before upsampling to dimensions 26x26. This feature map is then depth concatenated with the feature map from layer 61 to form a new feature map which is further fused with the 61st layer with the help of 1x1 convolution layers. The second detection layer is at the 94th layer with a 3D tensor of size 26x26x255.

For the final (third) detection layer, the same process is followed as that of the second detection where the feature map of the 91st layer is subjected to convolution layers before being depth concatenated and fused with a feature map from 36th layer. The final detection is made at the 106th layer with a feature map of size 52x52x255.

The multi-scale detector is used to ensure that the small objects are also being detected unlike in YOLO v2, where there was constant criticism regarding the same. Upsampled layers concatenated with the previous layers end up preserving the fine-grained features which help in detecting small objects.

The details of how this kernel looks in our model is described below in the next section.

5.4.2 Working of YOLOv3

The YOLOv3 network aims to predict bounding boxes (region of interest of the candidate object) of each object along with the probability of the class which the object belongs to.

For this, the model divides every input image into an SxS grid of cells, and each grid predicts B bounding boxes and C class probabilities of the objects whose centers fall inside the grid cells. The paper states that each bounding box may specialize in detecting a certain kind of object.

Bounding boxes “B” is associated with the number of anchors being used. Each bounding box has 5+C attributes, where ‘5’ refers to the five bounding box attributes:

- center coordinates (bx, by),
- height(bh),
- width(bw), and
- confidence score

and C is the number of classes.

Our output from passing this image into a forward pass convolution network is a 3-D tensor because we are working on an SxS image. The output looks like [S, S, B*(5+C)].

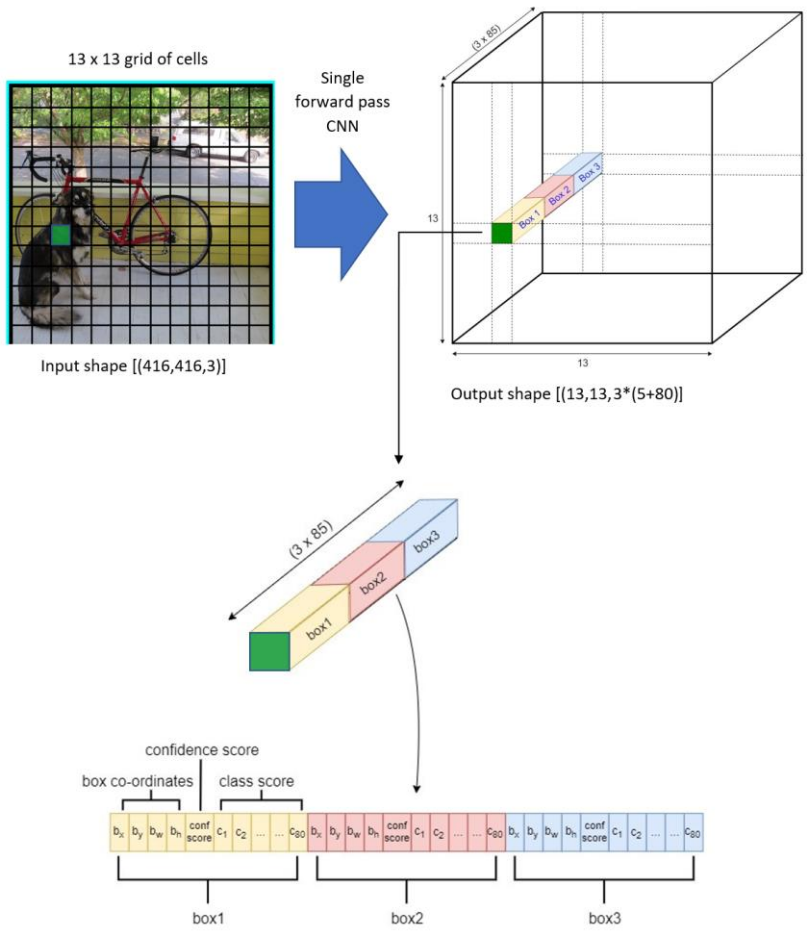


Figure 26: The basic principle of YOLOv3 [60]

In the above example shown in Figure 26, we see that our input image is divided into 13 x 13 grid cells. Now, let us understand what happens with taking just a single grid cell.

Due to the multi-scale detection feature of YOLO v3, a detection kernel of three different sizes is applied at three different places, hence the 3 boxes (i.e B=3). YOLOv3 was trained on the COCO dataset with 80 object categories or classes, hence C=80. Thus, the output is a 3-D tensor as mentioned earlier with dimensions (13, 13, 3*(80+5)).

5.4.3 Anchor Boxes

In the earlier years for detecting an object, scientists used the concept of the sliding window and ran an image classification algorithm on each window. Soon they realized this didn't make sense and was very inefficient, so they moved on to using CNNs and running the entire image in a single shot. Since the CNNs outputs square matrices of feature values (i.e something like a 13x13 or 26x26 in case of YOLO) the concept of "grid" came into the picture. We define the square feature matrix as a grid, but the real problem came when the objects to detect were not in square shapes. These objects could be in any shape (mostly rectangular). Thus, anchor boxes were started being used.

Anchor boxes are pre-defined boxes that have an aspect ratio set. These aspect ratios are defined beforehand even before training by running a K-means clustering on the entire dataset. These anchor boxes anchor to the grid cells and share the same centroid. YOLOv3 uses 3 anchor boxes for every detection scale, which makes it a total of 9 anchor boxes.

5.4.4 Non-Maximum Suppression

There is a chance that after the single forward pass, the output predicted would have multiple bounding boxes for the same object since the centroid would be the same, but we only need one bounding box which is best suited for all of them.

For this, we can use a method called non-maxim suppression (NMS) which basically cleans up after these detections. We can define a certain threshold that would act as a constraint for this NMS method where it would ignore all the other bounding boxes whose confidence is below the threshold mentioned, thus eliminating a few. But this wouldn't eliminate all, so the next step in the NMS would be implemented, i.e to arrange all the confidences of the bounding boxes in descending order and choose the one with the highest score as the most appropriate one for the object. Then we find all the other

boxes with high IoU with the bounding box with maximum confidence and eliminate all those as well.

5.4.5 Advantages of YOLOv3

There are major differences between YOLOv3, and older versions occur in terms of speed, precision, and specificity of classes.

YOLOv2 was using Darknet-19 as its backbone feature extractor, while YOLOv3 now uses Darknet-53. Darknet-53 is a backbone also made by the YOLO creators Joseph Redmon and Ali Farhadi. Darknet-53 has 53 convolutional layers instead of the previous 19, making it more powerful than Darknet-19 and more efficient than competing backbones (ResNet-101 or ResNet-152).

Backbone	Top-1	Top-5	Ops	BFLOP/s	FPS
Darknet-19	74.1	91.8	7.29	1246	**171**
ResNet-101	77.1	93.7	19.7	1039	53
ResNet-152	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Figure 27: Comparison of backbones. Accuracy, billions of operations (Ops), billion floating-point operations per second (BFLOP/s), and frames per second (FPS) for various networks [58]

Using the chart of Figure 28 provided in the YOLOv3 paper by Redmon and Farhadi [58], we can see that Darknet-52 is 1.5 times faster than ResNet101. The depicted accuracy doesn't entail any trade-off between accuracy and speed between Darknet backbones either since it is still as accurate as ResNet-152 yet two times faster.

YOLOv3 is fast and accurate in terms of mean average precision (mAP) and intersection over union (IOU) values as well. It runs significantly faster than other detection methods with comparable performance (hence the name You only look once).

Moreover, you can easily trade-off between speed and accuracy simply by changing the model's size, without the need for model retraining.

The chart in Figure 29, shows the average precision (AP) of detecting small, medium, and large images with various algorithms and backbones.

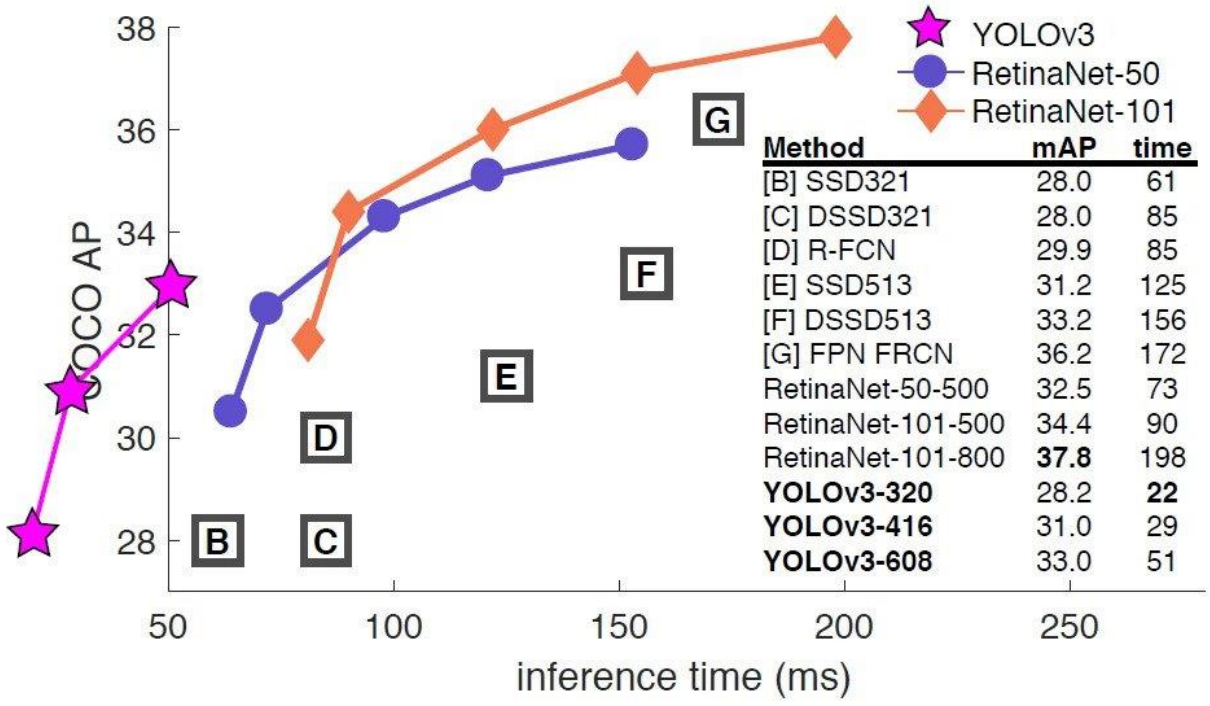


Figure 28: YOLOv3 runs much faster than other detection methods with a comparable performance using an M40/Titan X GPU [53]

The higher the AP, the more accurate it is for that variable. The precision for small objects in YOLOv2 was incomparable to other algorithms because of how inaccurate YOLO was at detecting small objects. With an AP of 5.0, it paled compared to other algorithms like RetinaNet (21.8) or SSD513 (10.2), which had the second-lowest AP for small objects. YOLOv3 increased the AP for small objects by 13.3, which is a massive advance from YOLOv2. However, the average precision (AP) for all objects (small, medium, large) is still less than RetinaNet.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI	Inception-ResNet-v2	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2	DarkNet-19	21.6	44.0	19.2	5.0	22.4	35.5
SSD513	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608×608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Figure 29: YOLOv3 comparison for different object sizes showing the average precision (AP) for AP-S (small object size), AP-M (medium object size), AP-L (large object size) [55]

Finally, the new YOLOv3 uses independent logistic classifiers and binary cross-entropy loss for the class predictions during training. These edits make it possible to use complex datasets such as Microsoft's Open Images Dataset (OID) for YOLOv3 model training. OID contains dozens of overlapping labels, such as "man" and "person" for images in the dataset.

YOLOv3 uses a multilabel approach which allows classes to be more specific and be multiple for individual bounding boxes. Meanwhile, YOLOv2 used a softmax, which is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector. Using a softmax makes it so that each bounding box can only belong to one class, which is sometimes not the case, especially with datasets like OID.

5.5 MS COCO Dataset

With applications such as object detection, segmentation, and captioning, the Microsoft's COCO dataset [62] is widely understood by state-of-the-art neural networks. Its versatility and multi-purpose scene variation serve best to train a computer vision model and benchmark its performance. The Common Object in Context (COCO) is one of the most popular large-scale labeled image datasets available for public use. It represents a handful of objects we encounter on a daily basis and contains image annotations in 80 categories, with over 1.5 million object instances.

Modern-day AI-driven solutions are still not capable of producing absolute accuracy in results, which comes down to the fact that the COCO dataset is a major benchmark for CV to train, test, polish, and refine models for faster scaling of the annotation pipeline. On top of that, the COCO dataset is a supplement to transfer learning, where the data used for one model serves as a starting point for another.

The COCO dataset is used for multiple CV tasks:

- Object detection and instance segmentation: COCO's bounding boxes and per-instance segmentation extend through 80 categories providing enough flexibility to play with scene variations and annotation types.
- Image captioning: the dataset contains around a half-million captions that describe over 330,000 images.

- Keypoints detection: COCO provides accessibility to over 200,000 images and 250,000 person instances labeled with keypoints.
- Panoptic segmentation: COCO's panoptic segmentation covers 91 stuff, and 80 thing classes to create coherent and complete scene segmentations that benefit the autonomous driving industry, augmented reality, and so on.
- Dense pose: it offers more than 39,000 images and 56,000 person instances labeled with manually annotated correspondences.
- Stuff image segmentation: per-pixel segmentation masks with 91 stuff categories are also provided by the dataset.

person	fire hydrant	elephant	skis	wine glass	broccoli	dining table	toaster
bicycle	stop sign	bear	snowboard	cup	carrot	toilet	sink
car	parking meter	zebra	sports ball	fork	hot dog	tv	refrigerator
motorcycle	bench	giraffe	kite	knife	pizza	laptop	book
airplane	bird	backpack	baseball bat	spoon	donut	mouse	clock
bus	cat	umbrella	baseball glove	bowl	cake	remote	vase
train	dog	handbag	skateboard	banana	chair	keyboard	scissors
truck	horse	tie	surfboard	apple	couch	cell phone	teddy bear
boat	sheep	suitcase	tennis racket	sandwich	potted plant	microwave	hair drier
traffic light	cow	frisbee	bottle	orange	bed	oven	toothbrush

Figure 30: COCO Classes [61]

For our project, we used a pretrained YOLOv3 model on MS COCO dataset. Configuration and weights file for training were obtained from the official YOLO algorithm page [53].

6. OUR HOLOLENS MIXED REALITY APPLICATION

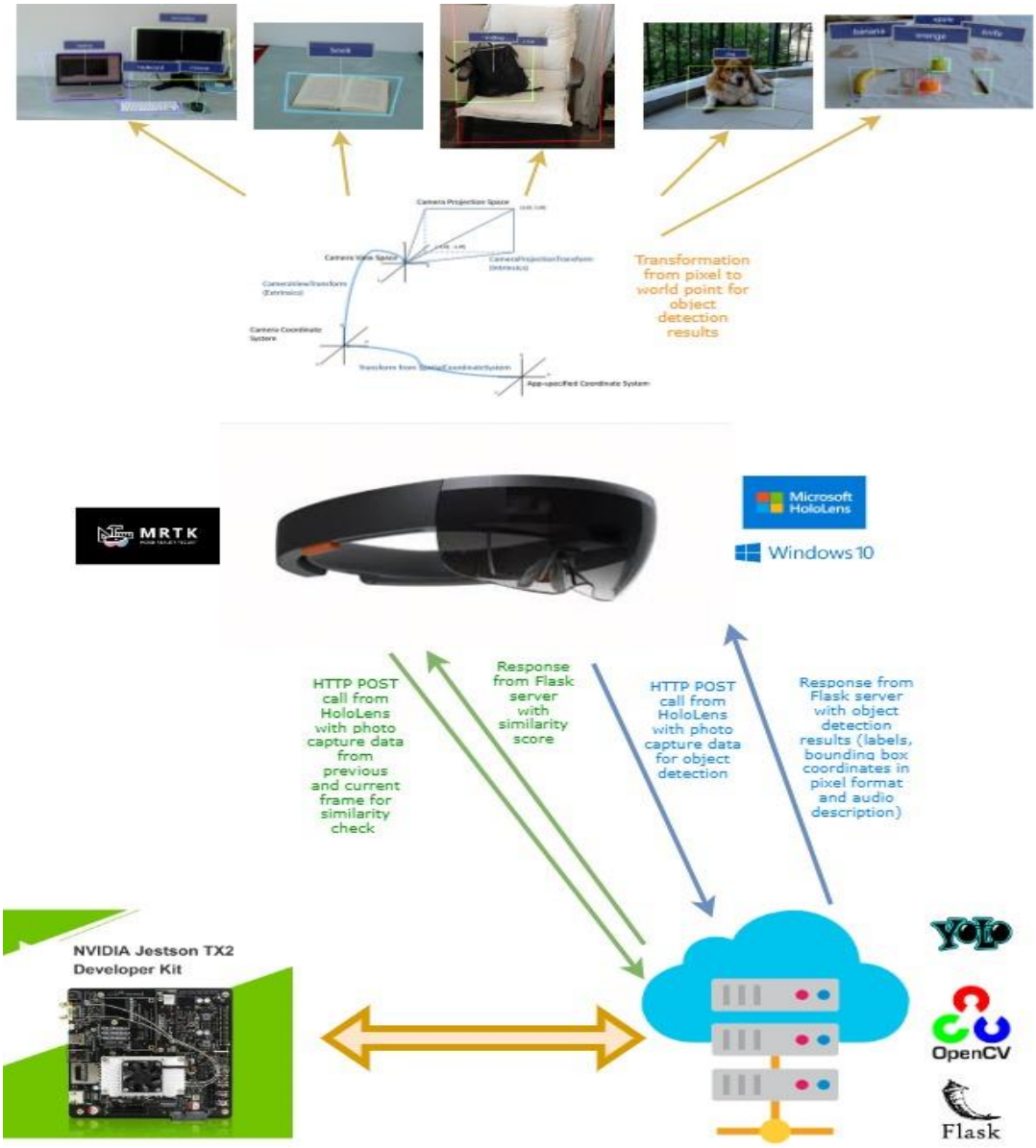


Figure 31: General Overview of the application

In the above figure, we give a general overview of the applications architecture. All the various components are coordinated by the application deployed to the HoloLens device.

6.1 General overview of the application

In this section we will describe the general architecture of our project.

The user is introduced in our HoloLens mixed reality application with a main menu scene. In this scene, the user can select to view the application's information with a comprehensive usage manual, or to initiate a visual or audio environment scan, all from a button collection. These main menu button options are selectable with a user's tap

gesture or alternatively, the user can speak a dedicated custom voice command for each option to activate it.

When the user enters the Visual or Audio scan scene, the MRTK Spatial awareness system is enabled to provide real-world environmental awareness. This is essential for placing our holograms for each object detection result.

Additionally, a green cursor follows the user while moving around, indicating that he/she can initiate the scan process. The environment scan is activated with a tap gesture or the voice command "Scan". Immediately after, the cursor becomes red as a visual indicator that a scanning is in process and results visualization is pending.

The first step is to perform a photo capture of the user's view. The photo capture frame obtained contains both the native image data and spatial matrices that indicate where the image was taken, which are going to be used after we receive the object detection results and help for their visualization. The second step is to perform a POST API call in order to send the native image data of the camera frame to our development server running on Jetson TX2 module and perform object detection on it. This task is accompanied by capture sound effect feedback, to ensure a better user experience, especially for users with vision impairment. As soon as a server response is available, we handle it respectively. In case of a potential server connection error or in case no objects are detected in the user's view, the scanning process is reinitiated, and the gaze cursor turns again to green. Otherwise, for each result, after transforming the 2D pixel coordinates in 3D world coordinates, we need to perform a raycast to the spatial mesh in order to find the collision point between this ray and the real scene and obtain the position for our holograms. For this task we user the spatial matrices obtained during the photo capture process. Each result is visualized with a tooltip enclosing the class name and a bounding box which surrounds the detected object. We also have sound feedback for both cases of a failure or success object detection response. Before placing any object detection result holograms in the view, we first clear all holograms from a past scan if any. After a successful scan, the gaze cursor turns green to inform the user that a new scan can be started.

For Audio scan scene, there are some differentiations regarding the functionality in compared to Visual scan scene. Firstly, the environment scan is started automatically when the user enters this scene and is repeated periodically each 30 seconds. Secondly, we compensate for cloud transmission and computation latencies by running camera frames similarity check between the current and previous HoloLens camera

capture frames, before applying the object detection algorithms on a camera frame, with purpose of avoiding running the object detection task when the user surrounding environment is significantly similar and limit as much as possible complex computations. This is achieved with a new API call to our server, where after performing a photo capture, we send the native image data from the current and the previous camera frames and obtain a similarity score based on the comparison of the correlation coefficient of the images' grayscale histograms. We proceed with an object detection API call to our server for the current camera frame only if the frames are not similar. For this API call, the request also includes a flag for returning an audio description of the detected results and their relative position in the user's view. Finally, using the HoloLens's spatial model, we also derive and announce distance information for each of the detected objects from the user.

We have established a custom voice command "Go to Menu", for the user to return to the application's main menu.

Our development server is running on NVIDIA Jetson TX2 module using Flask framework. We use a pretrained YOLOv3 model trained on MS COCO dataset. Object detection and frame similarity functions were developed with OpenCV library.

6.2 Setting up Unity Editor for Windows Mixed Reality Development

In order to deploy the application, we have to set all our tools with the correct settings.

Firstly, we will have to start with Unity. Installing the correct editor version is necessary. As mentioned in a previous chapter, we use Unity 2019.4.34f1. When starting Unity some work is done to generate files that are needed to start working on the project.

6.2.1 Project settings

Configuring Unity for Windows Mixed Reality development is a manual process, which must be completed whenever we create a new Unity project. Once our project is configured, our app will be able to do basic holographic rendering and spatial input. To target Windows Mixed Reality, the Unity project must be set to export as a Universal Windows Platform app. We also set the architecture to x86, as shown in Image 16.

Windows apps can contain two kinds of views, 2D views and immersive views. Apps can switch between their various immersive views and 2D views, showing their 2D views on a monitor as a window or in a headset as a slate [63].

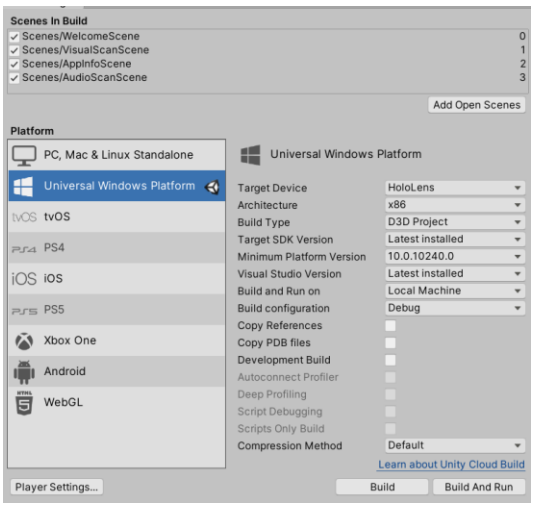


Image 16: Set up the project's platform in Unity

An immersive view gives our app the ability to create holograms in the world around us or immerse the user in a virtual environment. Apps that have at least one immersive view are categorized as mixed reality apps. Apps that never have an immersive view are 2D apps. In Unity, we can configure our project to create an immersive view by enabling Virtual Reality Supported. When virtual reality support is enabled, a virtual reality SDK must be added. As there is no separate SDK for Windows Mixed Reality development, the Windows 10 SDK is used instead.

In mixed reality apps, the scene is rendered twice, once for each eye to the user. This rendering method is referred to as stereoscopic vision. Compared to traditional 3D development, stereoscopic vision doubles the amount of work that needs to be computed. Therefore, it's important to select the most efficient rendering path in Unity to save both on CPU and GPU time. Single pass instanced rendering optimizes the Unity rendering pipeline for mixed reality apps and therefore it's recommended to enable this setting by default for our project.

To achieve better hologram stability from the perception of the user, Depth Buffer Sharing should also be enabled. By turning this on, Unity will share the depth map produced by your app with the Windows Mixed Reality platform. The platform will then be able to better optimize hologram stability specifically for our scenes for any given frame being rendered by our app. With regards to performance, selecting the 16-bit depth format compared to 24-bit will significantly reduce the bandwidth requirements as less data will need to be moved/processed.

All these settings can be configured from Unity’s player settings, as shown in Image 17.

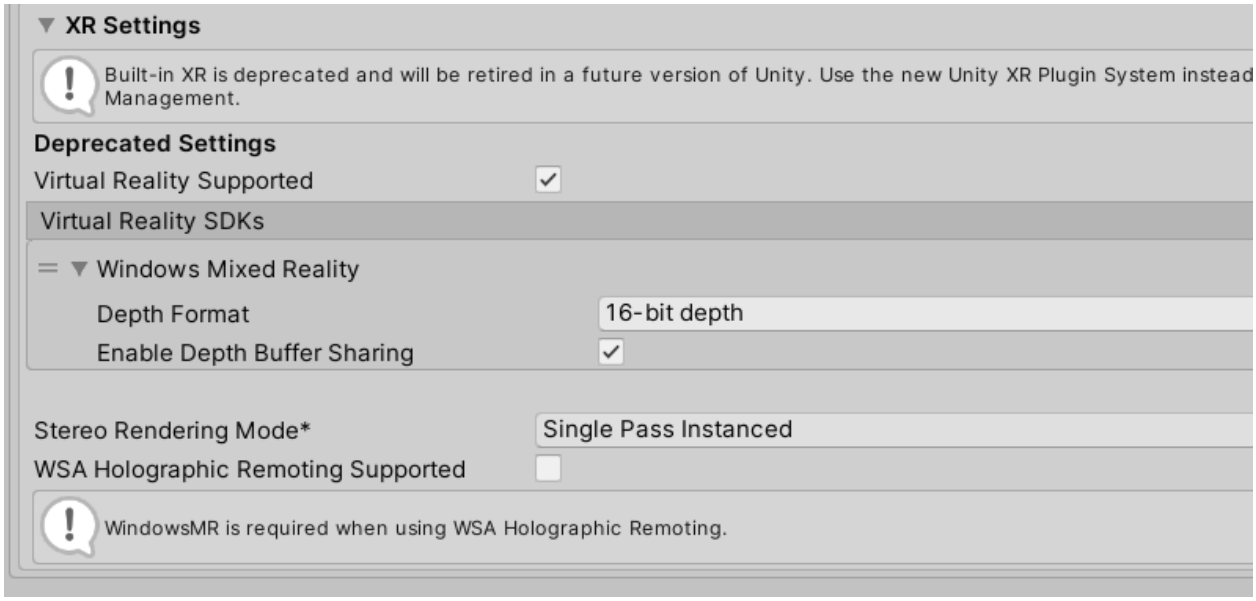


Image 17: Setting up Unity's XR Settings for our app

6.2.2 MRTK import

The next step is to load all necessary libraries and projects we need to create our application. This is the Mixed Reality Toolkit and some other assets that are loaded from Unity. The toolkit also sets some configuration options that have to be set in order to launch the application and deploy it to the device. The steps we need to follow are:

- Download and import MRTK foundation package from the official documentation page [64]
- Follow the steps found with the above link
- Import TMP Essentials Resources, and restart Unity to see changes.

One of the main ways that MRTK is configured is through the many profiles available in the Foundation package. Profiles configure the behavior of MRTK core components. The MixedRealityToolkit object contains the active profile and can be viewed in the Inspector window. When MRTK is added to the scene, the preselected profile is DefaultMixedRealityToolkitConfigurationProfile. The default MRTK profiles can't be modified. However, since we had to add custom voice commands for our application, we had to clone this profile and modify the Input settings appropriately.

6.2.3 Application permissions

Our application also requires some extra user permissions based on the functionalities that are available. These permissions are enabled from Unity’s Capabilities section under Player settings. Since our application need to have access to the HoloLens

camera we have to enable “WebCam”. For the user’s input, we need “GazeInput” for gaze and “Microphone” for voice commands. We also enabled “SpatialPerception” for spatial awareness system. Finally, we enabled “InternetClient” for our API calls started from HoloLens app to our development server.

6.3 Scripts

In order to create our application and also implement some logic we had to write some code. For the scripts we have created we used the C# programming language and some scripting APIs. We used the Unity scripting API which offers the documentation and some basic examples for the use of functions and classes Unity offers. This scripting API is the core of our application and there were no alternatives to use. We used the .NET framework from Microsoft that was used to build and deploy our application to the HoloLens device.

Some basic components for our application are created and enabled as GameOjects in Unity and we have added basic scripts to them. This is a common practice when developing apps in Unity. For instance, there is an object that is called SceneChanger and attached to it is a script that handles the scene changing functionality.

6.3.1 Script structure & Execution order

Unity creates scripts with use of a default template. According to the template the script has the name of the public class that resides in it. Two default functions are also generated and are important in order to understand a script’s lifecycle. It is not strict to use any of these methods, but they indicate some key features of a Unity application.

The two methods are the Start method and the Update method. The first one gets called automatically and is used mainly for initialization purposed. The second one, the Update method is very important if used. The Update method is called in every frame. The CPU scheduler keeps track of these functions and calls them in every frame. We mention these methods because the code or anything a programmer might call in the methods body has a great impact on the overall performance of the application. We investigate this fact in the Performance chapter. We present the default template of a unity script from our application’s gaze cursor in Image 18.

```
public class GazeCursor : MonoBehaviour
{

    // The cursor (this object) mesh renderer
    private MeshRenderer meshRenderer;

    // Runs at initialization right after the Awake method
    Unity Message | 0 references
    void Start()
    {
        // Grab the mesh renderer that is on the same object as this script.
        meshRenderer = gameObject.GetComponent<MeshRenderer>();

        // Set the gaze cursor reference in scene organiser
        ObjectDetectionSceneOrganiser.Instance.gazeCursor = gameObject;
        gameObject.GetComponent<Renderer>().material.color = Color.green;

        // Change the size of the cursor
        gameObject.transform.localScale = new Vector3(0.01f, 0.01f, 0.01f);
    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        // Do a raycast into the world based on the user's head position and orientation.
        Vector3 headPosition = Camera.main.transform.position;
        Vector3 gazeDirection = Camera.main.transform.forward;
```

Image 18: Default Unity script template as used in our app's gaze cursor

6.3.1 Coroutines

Coroutines are a type of functions but with some more and very important characteristics.

In general, functions are called and executed inside the lifecycle of a frame. This means that a function has to run to its completion, and this had to happen in a single frame. Also, many times it is needed to fulfill a task that is going to take multiple frames to complete. We could spread this work across multiple frames with the use of the Update function, but this solution will certainly backfire on us with performance issues. There are many tasks that we know for sure that will take some time and that some procedures should not start or execute upon the completion of a specific task. For example, in our application we use some REST calls that we want to wait and get the server response. Unity offers a solution to our problem, which are the coroutines.

Coroutines are like functions but have the ability to pause their execution and return the control to the main thread of the application. This is very useful when waiting for input from any source, in our case the source is the network, and then return control the coroutine to start where it left off and terminate its execution [65].

In order to return the control from a coroutine to main thread a specific instruction is used, the yield instruction followed by some condition that, if met, the control of the application's execution flow should return to the coroutine.

Coroutines might also be used for very large operations, even infinite loops that do some useful work and we want to operate for the application's whole lifecycle. If we implemented this with any common function or the Update function the application would have non expected behaviors and results. Also, there is the option to stop or even pause a coroutine and start it again from the point it left off and continue whatever operation was intended to be completed. To permanently stop a coroutine we can also destroy it.

6.4 Application main menu

When the user selects our HoloLens application from the applications menu, he is presented with our app's custom spash screen, as seen in Image 19 below. We can set a custom app icon and spash screen from Unity's Player settings, under the Icon and Splash Image sections respectively.

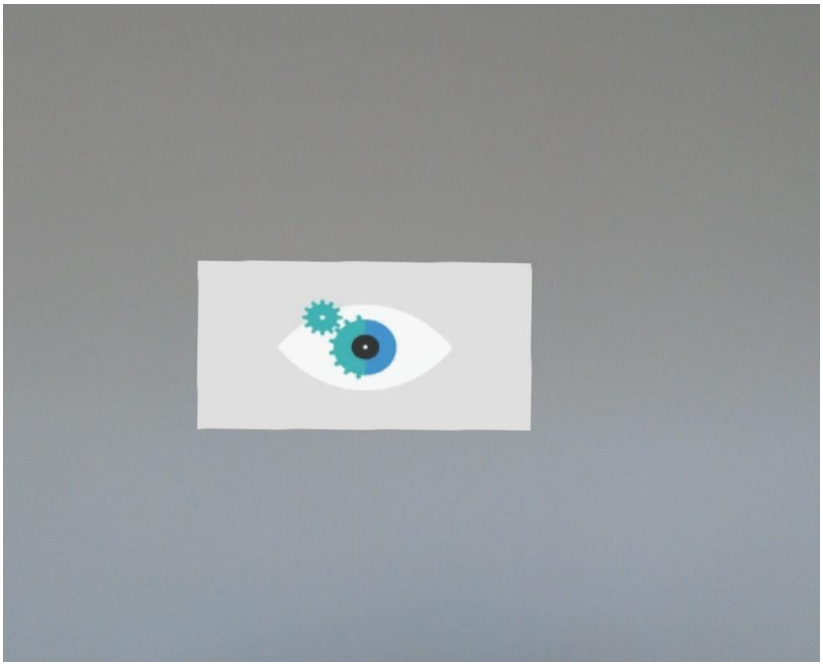


Image 19: Our HoloLens application splash screen

When our application launches, the user is introduced in our app with a main menu depicted with a button collection, where three options are available, a "Visual scan" option for an object detection environment scan with visual only feedback, an "Audio scan" option for an audiovisual scan which is a suitable choice for users with visual impairments, and finally with an option "App info", where the user can see a panel with a

comprehensive guide for using our app and general information about its components and implementation.

Image 20: Our HoloLens application main menu

The application's main menu is constructed from WelcomeScene in our Unity project. The scene's structure and game objects hierarchy are presented in Image 21.

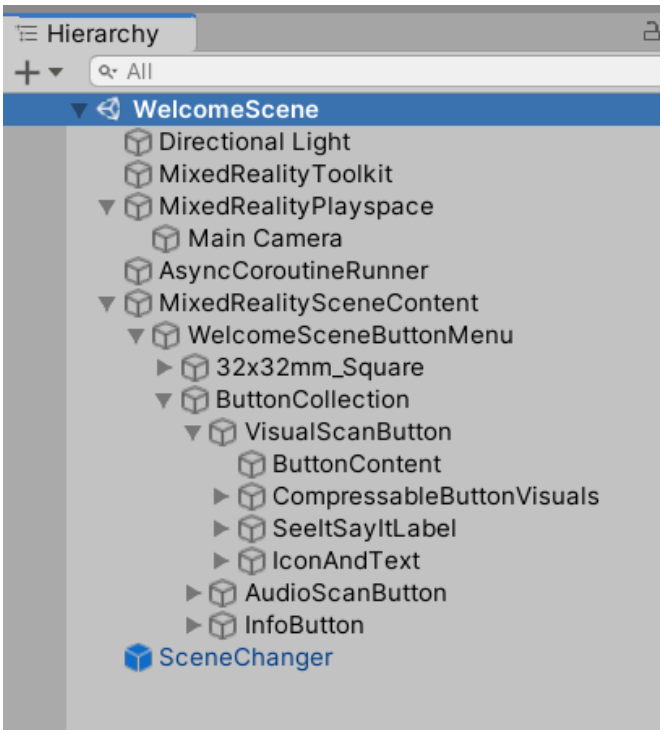


Image 21: Unity WelcomeScene hierarchy

The objects in this scene are, besides the lighting and the imported MRTK mandatory objects (Directional Light, MixedRealityToolkit, MixedRealityPlayspace), are the following:

- WelcomeSceneButtonMenu, which is a Horizontal HoloLens button bar with shared backplate containing 3 buttons, taken as a prefab from MRTK foundation package [66]
- SceneChanger prefab, used for user navigation.

Unity's Prefab system allows us to create, configure, and store a GameObject complete with all its components, property values, and child GameObjects as a reusable Asset. The Prefab Asset acts as a template from which we can create new Prefab instances in our Scenes. When we want to reuse a GameObject configured in a particular way in multiple places in a Scene, or across multiple Scenes in our Project, we should convert it to a Prefab. This is better than simply copying and pasting the GameObject, because the Prefab system allows us to automatically keep all the copies in sync. Any edits that we make to a Prefab Asset are automatically reflected in the instances of that Prefab, allowing us to easily make broad changes across our whole project without having to repeatedly make the same edit to every copy of the Asset.

Regarding the SceneChanger prefab, is an empty game object where we have attached a script for handling the user's navigation according to the main menu button that is selected.

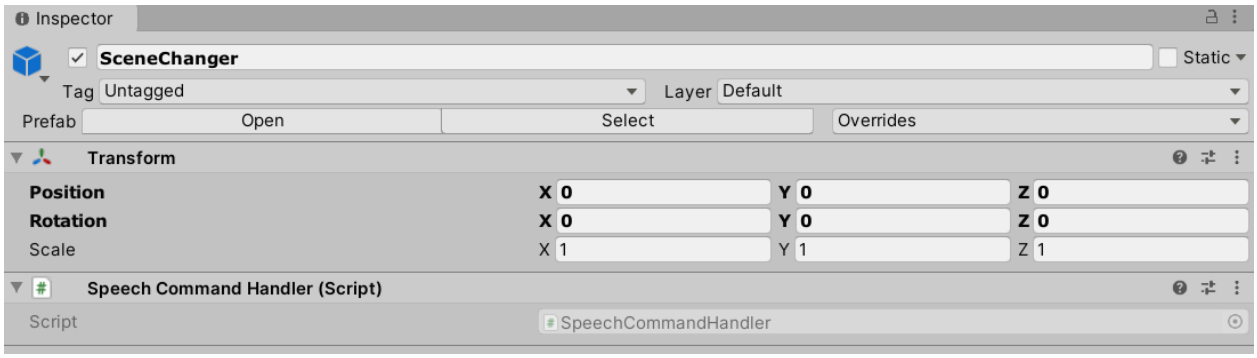


Image 22: SceneChanger prefab asset

In SpeechCommandHandler script, we have added a function with name "ChangeScene", which loads a specific Unity scene given its name, in order for the user to navigate from the current scene to a new one. Function "CaptureImageForAnalysis"

is used for capturing an image for object detection with “Scan” voice command and will be described in detail in a following section.

```
using UnityEngine;
using UnityEngine.SceneManagement;

Unity Script | – references
public class SpeechCommandHandler : MonoBehaviour
{
    – references
    public void ChangeScene(string sceneName) => SceneManager.LoadScene(sceneName);

    – references
    public void CaptureImageForAnalysis()
    {
        // Only if the object detection process does not start automatically
        if (!ObjectDetectionSceneOrganiser.Instance.objectDetectionProcessIsAutomatic)
        {
            ImageCapture.Instance.StartImageCaptureProcess();
        }
    }
}
```

Image 23: SpeechCommandHandler script

As we have mentioned before, the main menu buttons can be triggered with a custom voice command apart from the user’s tap gesture which is handled from MRTK. Since the default MRTK profiles can't be modified, in order to register our custom voice commands for our application, we had to clone the default MRTK profile DefaultMixedRealityToolkitConfigurationProfile, and create our custom one, named CustomMixedRealityToolkitConfigurationProfile. Voice commands are configured under the Input section of our profile. For enabling the edit button for voice commands, we also had to clone the default input system profile and create a custom one without further changes apart from our new voice commands. Our custom mixed reality configuration profile and voice commands are presented in detail in the following Images 24 and 25 respectively. We should note that this custom mixed reality configuration profile is used throughout our app and not solely for this specific WelcomeScene for our main application menu.

We should now proceed with describing in detail our main menu button collection and its configuration for supporting the user’s navigation through our app, which is constructed by WelcomeSceneButtonMenu, under MixedRealitySceneContent. This horizontal button collection is offered as a button prefab from MRTK foundation package and consists of 3 buttons, which can be found under ButtonCollection game object. These are VisualScanButton, AudioScanButton and AppInfoButton.

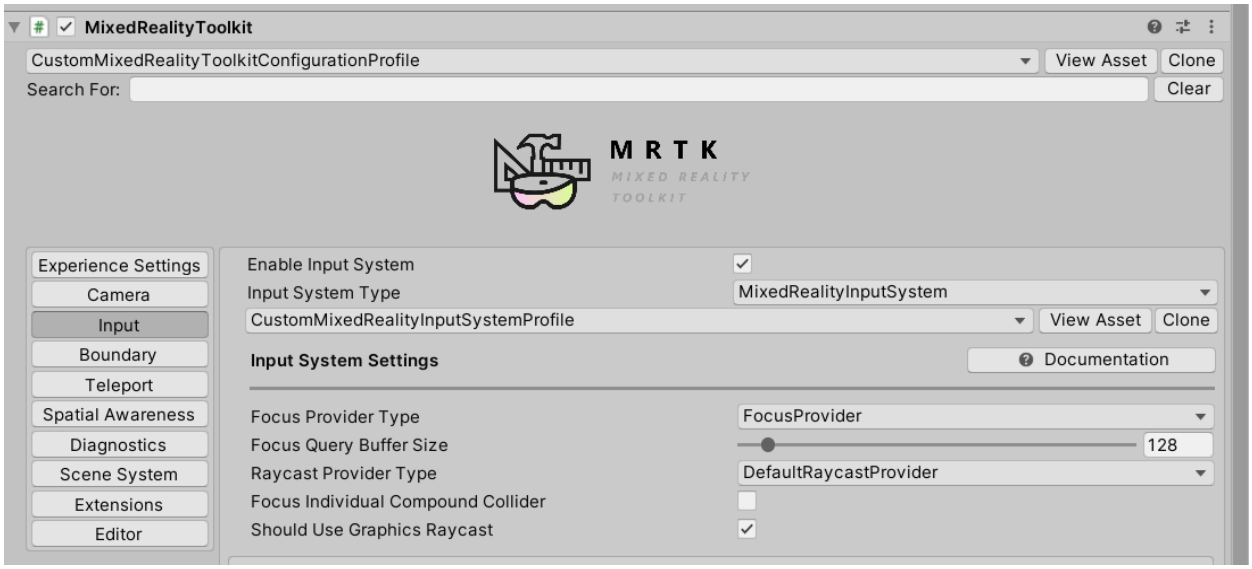


Image 24: Custom Mixed Reality Configuration profile for our app

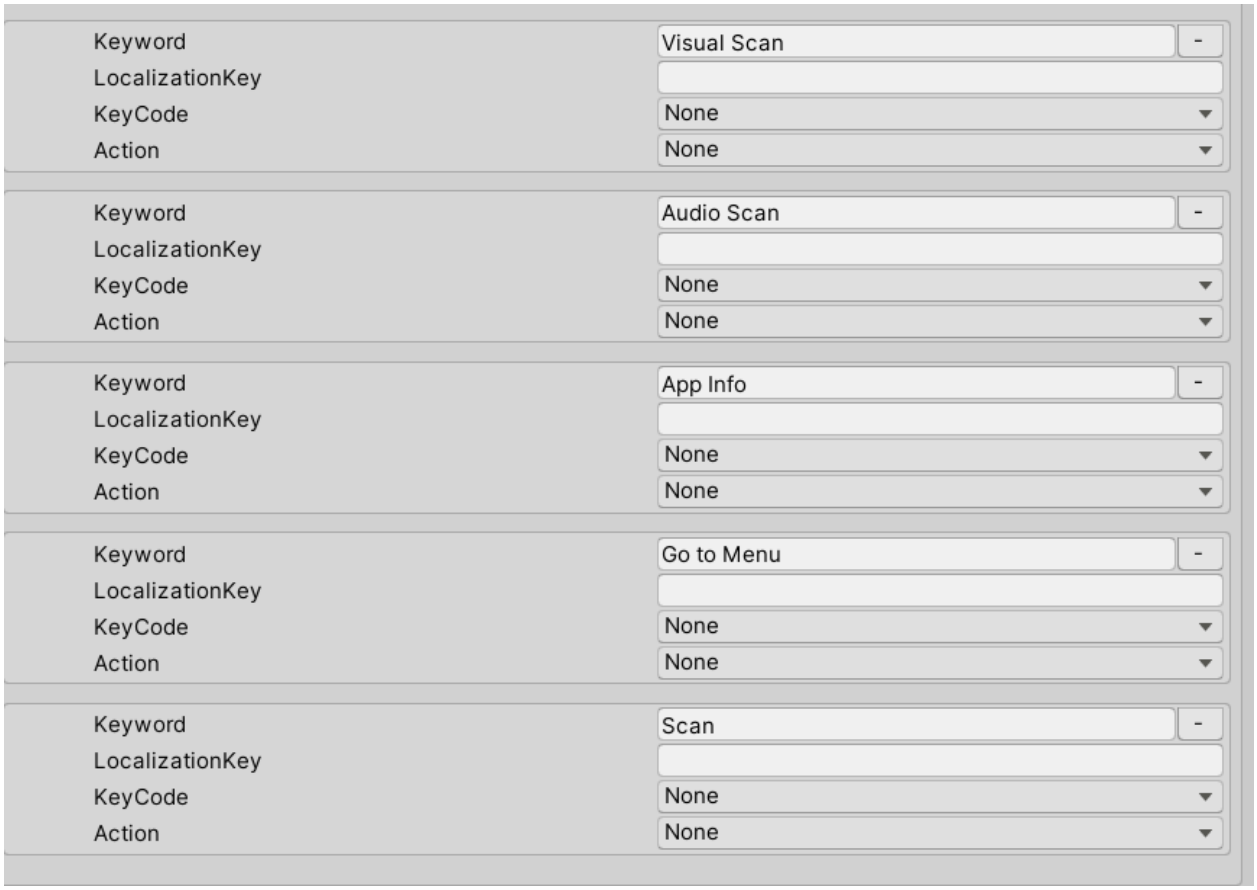


Image 25: Our application's custom voice commands

To enhance user experience and immerse, we have made WelcomeSceneButtonMenu to follow the user's view, so that the user will not lose sight of the main menu regardless of any type of movement he commits. As we have mentioned previously in section 4.7, we achieve this functionality with adding a Follow class solver [48] to our button

collection WelcomeSceneButtonMenu. After some time, the UI follows the user's movement and view tracking any movement in any direction. For example, if the user lifts his head up, after some time the main menu will follow his view. Our configuration for this purpose is depicted in Image 26 below, and the result of the main menu following the user's movements can be seen in Image 14.

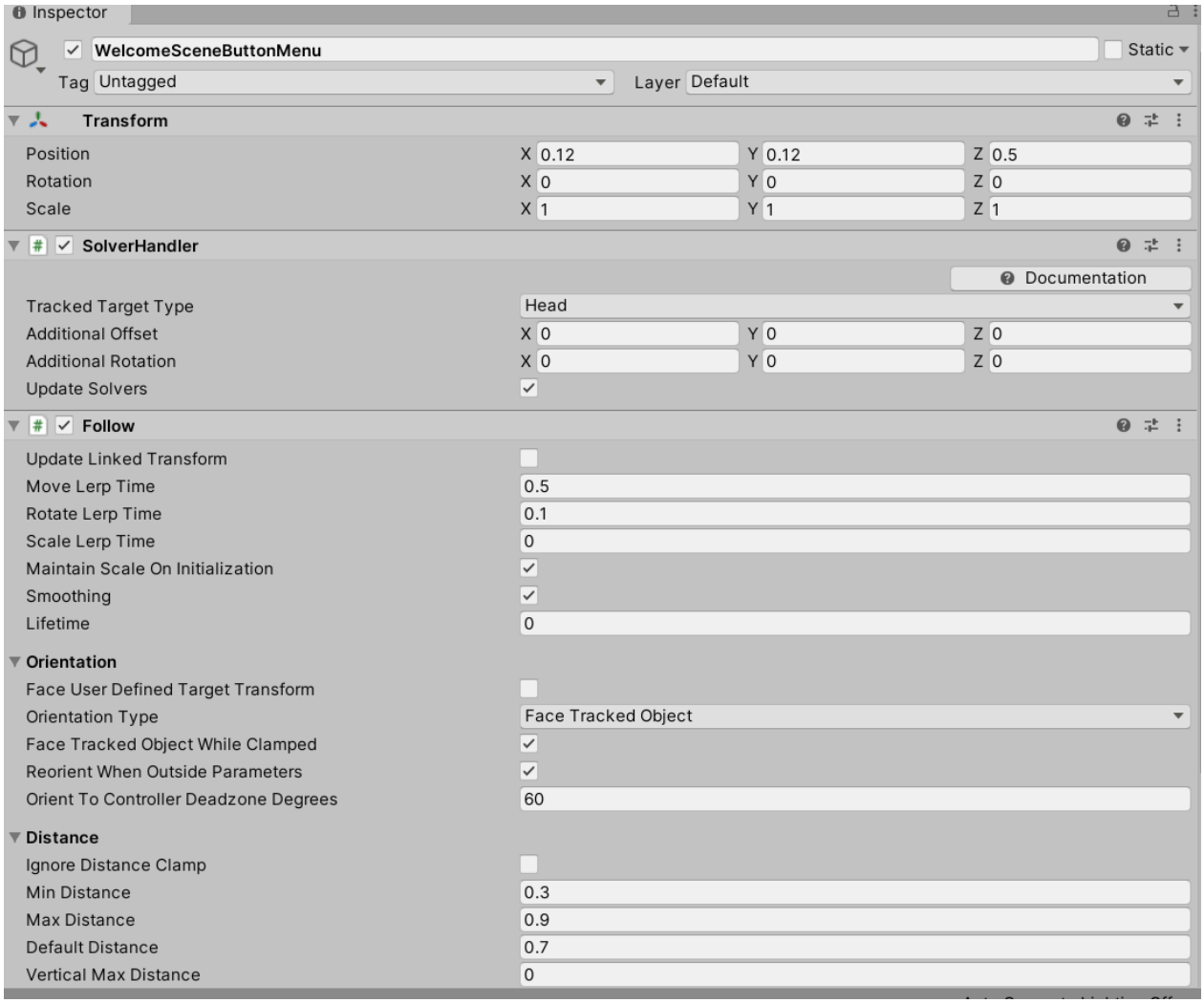


Image 26: Follow solver for our application main menu button collection

Finally, we should describe in detail the configuration for our buttons. Since our buttons are configured in the same way, and the only differences concern the voice commands and the Unity scene names, we will use the VisualScanButton as a reference. In Image 27, we can see the configurations regarding the label and the click event. We have enabled the main label switch to display a button label and set its value to "Visual Scan". We have also enabled the "See it, Say it label" and "Display speech command" switches, thus when the user is gazing to the button, a label "Say Visual Scan" we be

displayed below the button, to inform the user for the speech command he can use to activate the button. The action performed when the user selects the button with speech command or with tap gesture, is configured in the On Click event section. As we can see, we have set to call the “ChangeScene” function that is available in SpeechCommandHandler script, and the given argument for loading the Visual scan scene is “VisualScanScene”, which equals to the scene’s Unity name. The passed arguments for audio scan and application info buttons are “AudioScanScene” and “AppInfoScene” respectively.

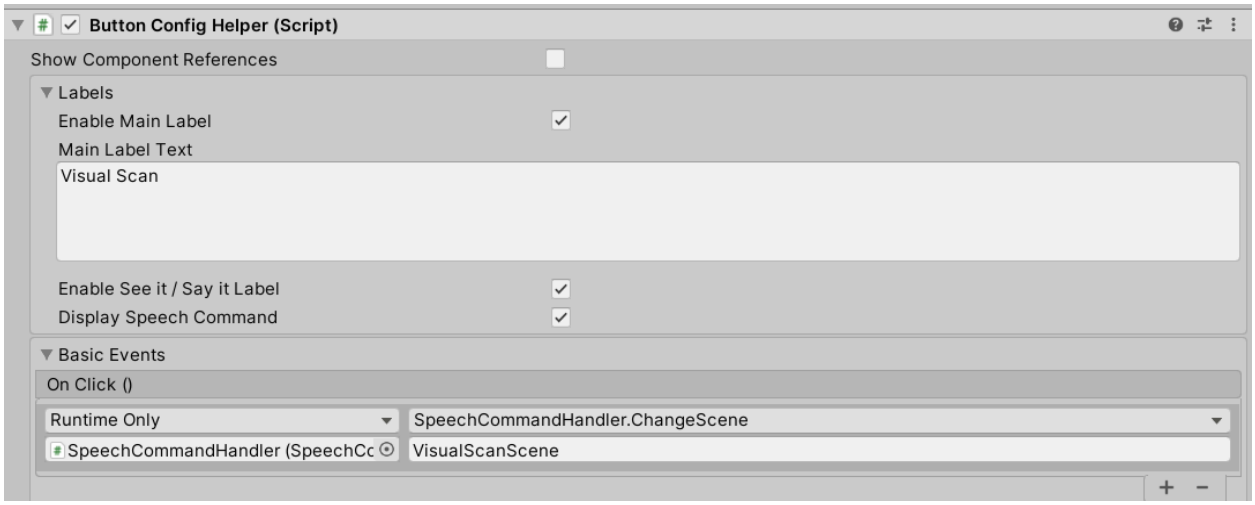


Image 27: Visual scan button configuration for the label and click event

As seen in Image 28, we set that the voice command that will trigger our button is “Visual scan”, selected from our application’s voice command drop-down list. We have not selected the “Requires Focus” option, so that the user can use the voice command and trigger the button selection, without the need to gaze at it, taking under consideration that a user with visual impairments can select the button with a voice command seamlessly.

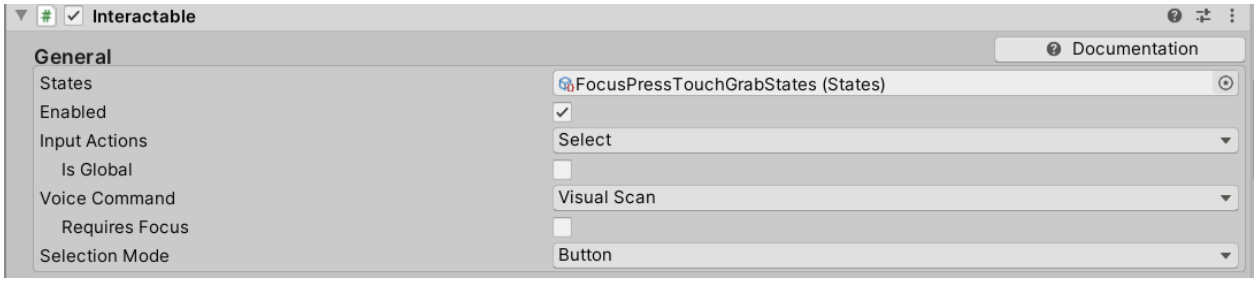


Image 28: Visual scan button configuration for voice command

Finally, as seen in Image 29, we can select the button's icon from a set of default ones.

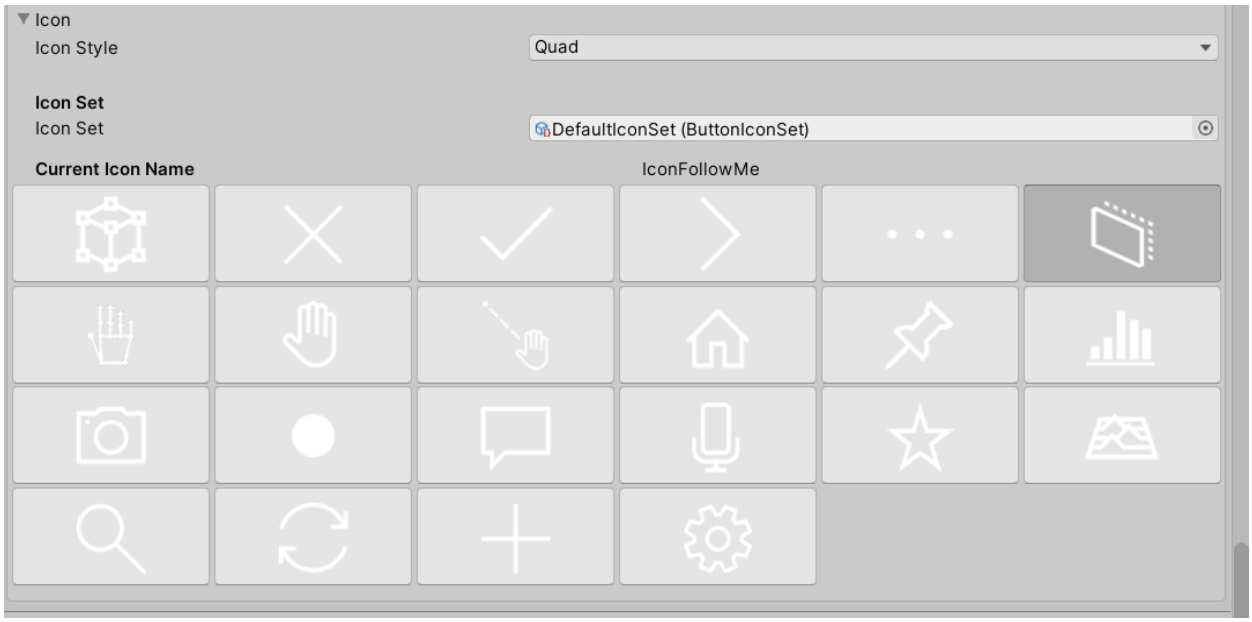


Image 29: Visual scan button icon configuration

Images 30, 31 and 32 depict our application's main menu buttons, while the user is gazing at each of them, and the "See it, Say it" correspondent label with the voice command is shown for each of them.

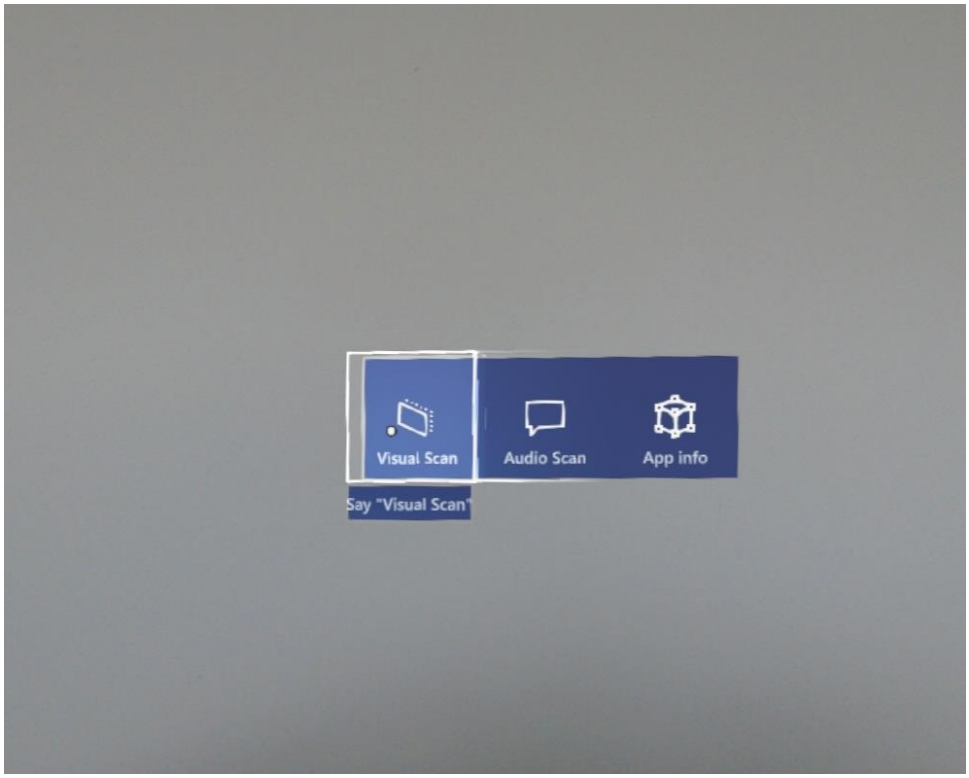


Image 30: Visual scan button during user gaze with voice command label

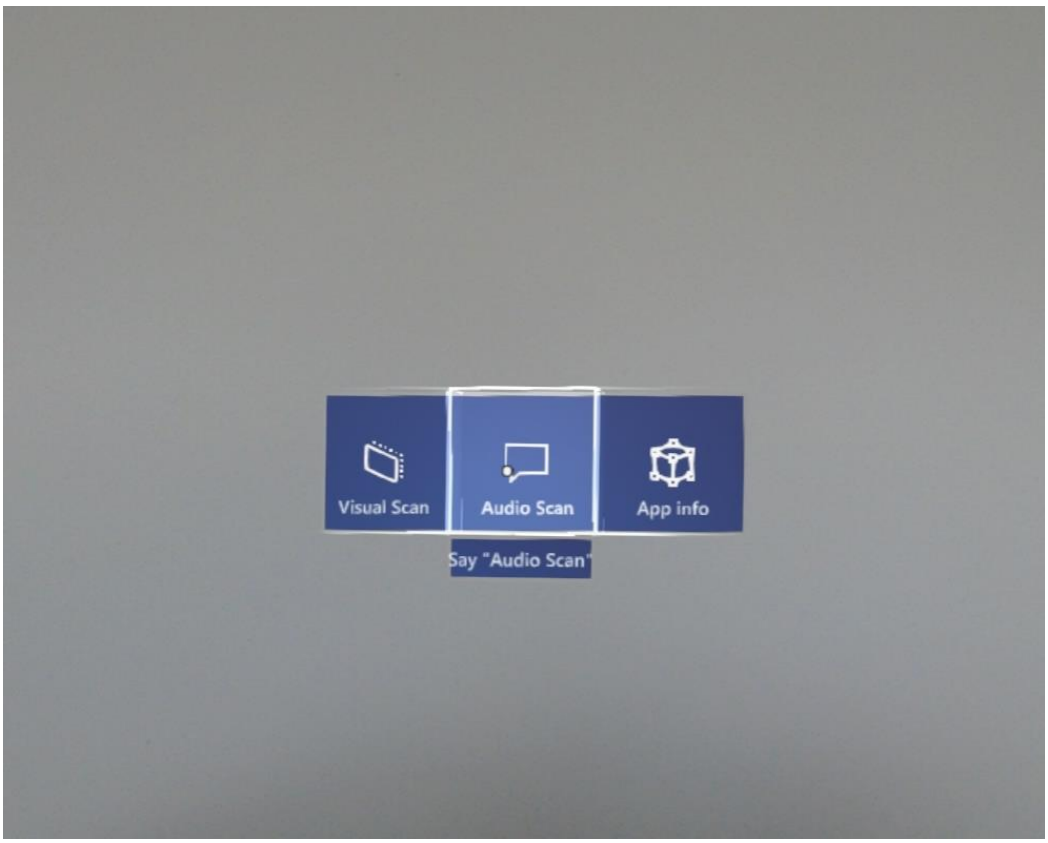


Image 31: Audio scan button during user gaze with voice command label

Image 32: App Info button during user gaze with voice command label

6.5 Application Information scene

In case the user selects the “App Info” button, either with a tap gesture or a voice command, he is navigated to the App Info scene, where a description panel with information about the app’s component and usage is present, as depicted in Image 33.

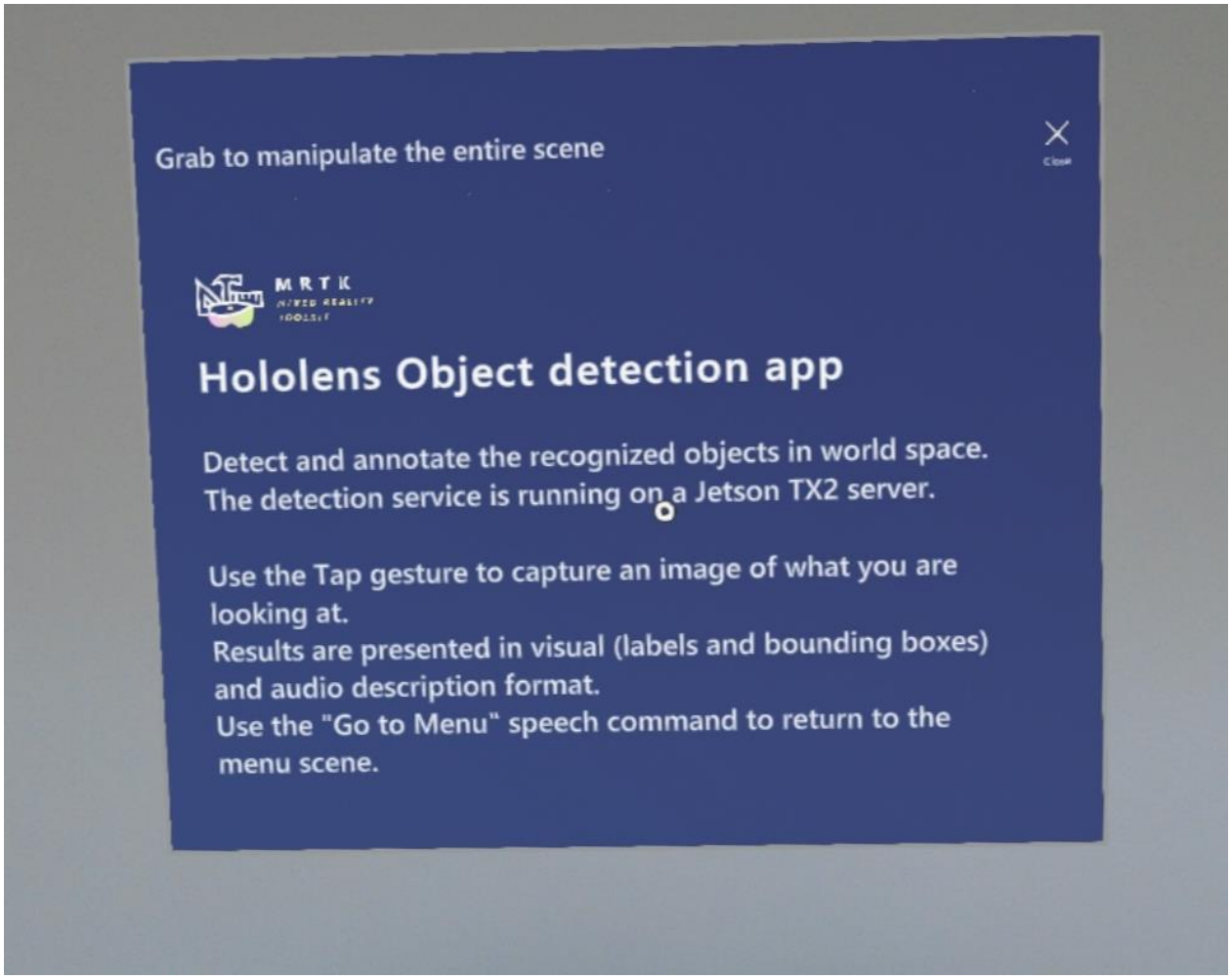


Image 33: Our application's information panel in App Info scene

This panel also supports following the user’s movements based on Follow toggle solver, configured in the same way as described for our application’s main menu button collection in WelcomeScene.

The application’s info panel is constructed from AppInfoScene in our Unity project. The scene’s structure and game objects hierarchy are presented in Image 34. The objects in this scene include, besides the lighting and the imported MRTK mandatory objects (Directional Light, MixedRealityToolkit, MixedRealityPlayspace), our description panel, which is located under the MixedRealitySceneContent game object and is constructed from SceneDescriptionPanelRev. This panel is not part of MRTK foundation package

prefab assets, but is used in MRTK Examples Hub project to describe the contents and functionality of each scene [67], so we used this for a similar purpose in our app. The user can grab the panel to manipulate the entire scene and place it in a specific point in his view.

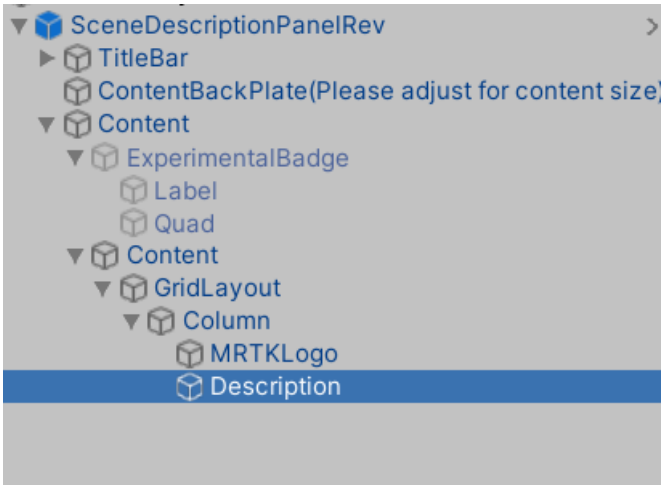


Image 34: Unity AppInfoScene hierarchy

The SceneDescriptionPanelRev consists of a title bar for the grab action message and a content grid layout, for the MRTK logo and the app description text. As seen in Image 35, we use a different text size and style to distinct the description header from the information text.

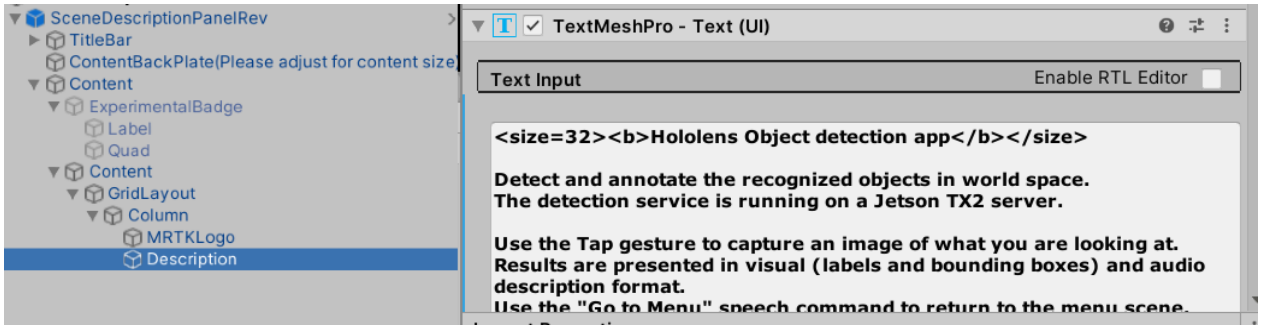


Image 35: App Info panel description configuration

The user can navigate back to the main menu scene using our custom voice command “Go to menu”. This command has been added to the input system of our custom MRTK configuration profile. According to MRTK documentation, The Speech Input Handler script can be added to a GameObject to handle speech commands using UnityEvents. This script automatically shows the list of the defined keywords from the Speech Commands Profile, and it is up to the developer to handle appropriately according to the app needs the voice command [68]. In our case, we have added the Speech Input Handler script to our root MixedRealitySceneContent game object. As depicted in Image

36, we have selected to handle “Go to Menu” speech command, where user focus is not required, and when the command is triggered, we use again our aforementioned function “ChangeScene” from SpeechCommandHandler script, in order to load our WelcomeScene and navigate the user back to the main menu.

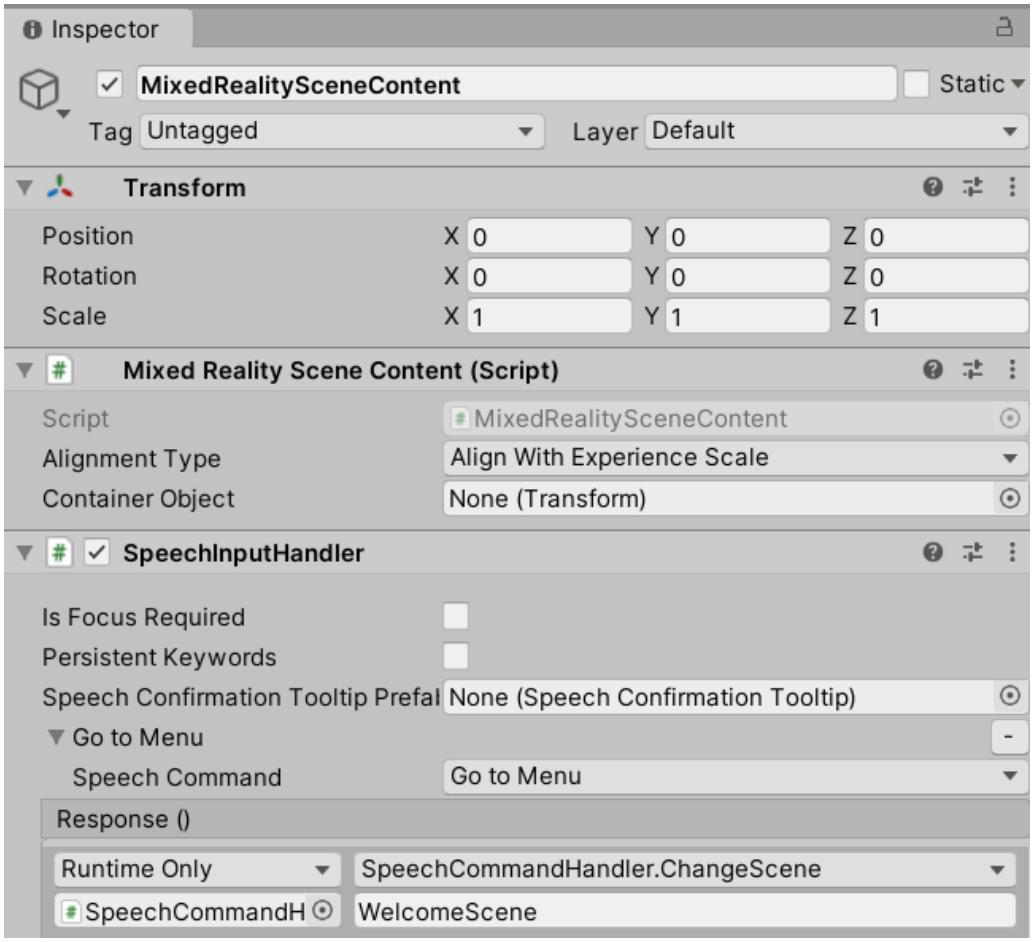


Image 36: Speech Input Handler configuration to navigate the user back to main menu from app info scene

6.6 Visual and Audio scan scenes

6.6.1 Scanning operation steps

When the user selects the Visual or Audio scan option from our main menu, he is redirected to the respective scan scene, where upon entering he can see a gaze cursor. The cursor is green when our application is not busy performing background operations related to an object detection scan, and red while a scanning process is in progress and the results rendering is pending. When the gaze cursor is in green state, the user can initiate an object detection scan with a tap gesture or with the voice command “Scan” for

Visual scan, whilst for Audio scan the process starts automatically and is repeated. As long as the gaze cursor is in red state, tap gestures and voice commands will not trigger a new scan until the already started one finishes and its results are rendered in the user's view. The main operation steps of an object detection scan using our app are the following:

- Upon tap gesture, voice command or automatic scan start, the gaze cursor is set to red state, and we perform a camera capture of the user's view. The camera capture object is a combination of native image data and spatial info matrices that indicate where the image was taken.
- We then perform a POST API call to our development server sending the native image data as a request parameter for object detection to be performed on the camera capture image. For Audio scan, this step is performed without further checks only the first time of the periodic scans. Otherwise, we firstly perform a POST API call to our development server with the native image data for the current and previous camera capture frames, and if the similarity between them is below a threshold defined server side, we continue to the POST API call for object detection, adding a flag for also returning an audio description text of the results.
- Upon getting a response from our server, in case of error the scan process is reinitialized. In case of success, the response retrieved from server has to be transformed before rendering.
- The transformation of the API to our application's domain model consists of transforming the center and the four vertices of the bounding box for each of the results from 2D pixel coordinate system to 3D world coordinate system, and then we need to perform a raycast to the spatial mesh for each world point in order to find the collision point between this ray and the real scene to obtain the position for our holograms. For audio scan, the audio description text returned from server is enriched with distances information from the user.
- After successful rendering of our holograms (a tooltip and bounding box for each result) and audio description announcement if available, the scan process is reinitialized, and the cursor is set again to green state for a new scan to start upon user's input or automatically.

6.6.2 Scanning views hierarchy and component structure

In this section we will describe the hierarchy and components structure for the Visual and Audio Unity scenes in our project. As depicted in Image 37, the objects in this scene, besides the lighting and the imported MRTK mandatory objects (Directional Light, MixedRealityToolkit, MixedRealityPlayspace), are the following:

- The GazeCursor prefab asset, attached to the scene's main camera game object, which is the cursor used in our application used to visualize the object detection process state to the user. It works together with the SpatialMapping prefab to be able to be placed in the scene on top of physical objects and follow the user's movement.
- The SpatialMapping prefab asset, attached to the scene's main camera game object, which is the object that enables the scenes to create and use a virtual map, using the Microsoft HoloLens' spatial tracking.
- The AudioEffectsPlayer prefab asset, attached to the scene's main camera game object, used for playing certain sound effects when the object detection process starts and finishes, with a different sound effect based on the result status (success or failure)
- The PredictionObjectsContainer for VisualScanScene and Audio PredictionObjectsContainer for AudioScanScene, attached to MixedRealitySceneContent game object, used as container game objects to attach our object detection results holograms (tooltips and bounding boxes), which are created dynamically from our scripts.

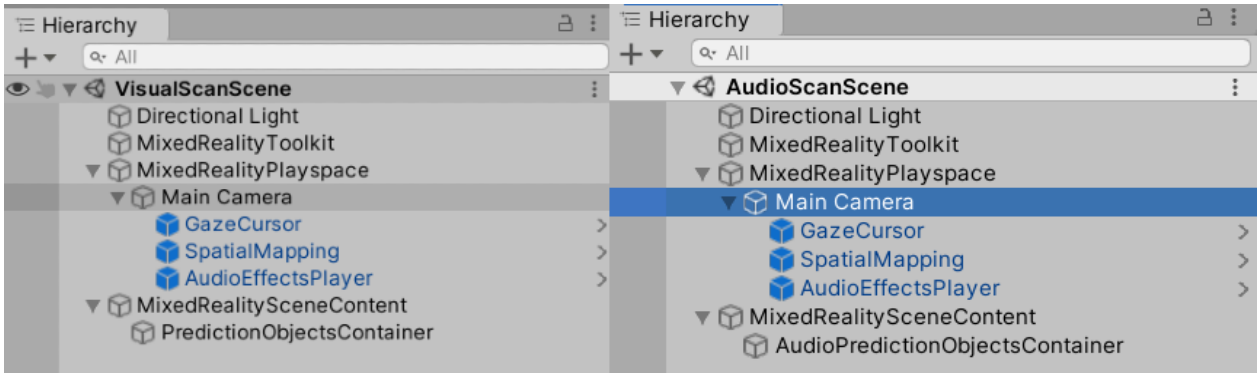


Image 37: Hierarchy for Visual and Audio scan scenes in our project

For MixedRealitySceneContent game object, we have added MRTK Speech Input Handler script, to handle "Go to Menu" voice command in order to navigate the user back to the main menu, as described previously for AppInfo scene, and additionally to

handle voice command “Scan” for initiating the object detection scan process. In this case, we call the function “CaptureImageForAnalysis” from SpeechCommandHandler script, as shown in Image 23, where after checking if the object detection process does not start automatically, we call the function to start the image analysis from ImageCapture script, which is described in detail in a following section.

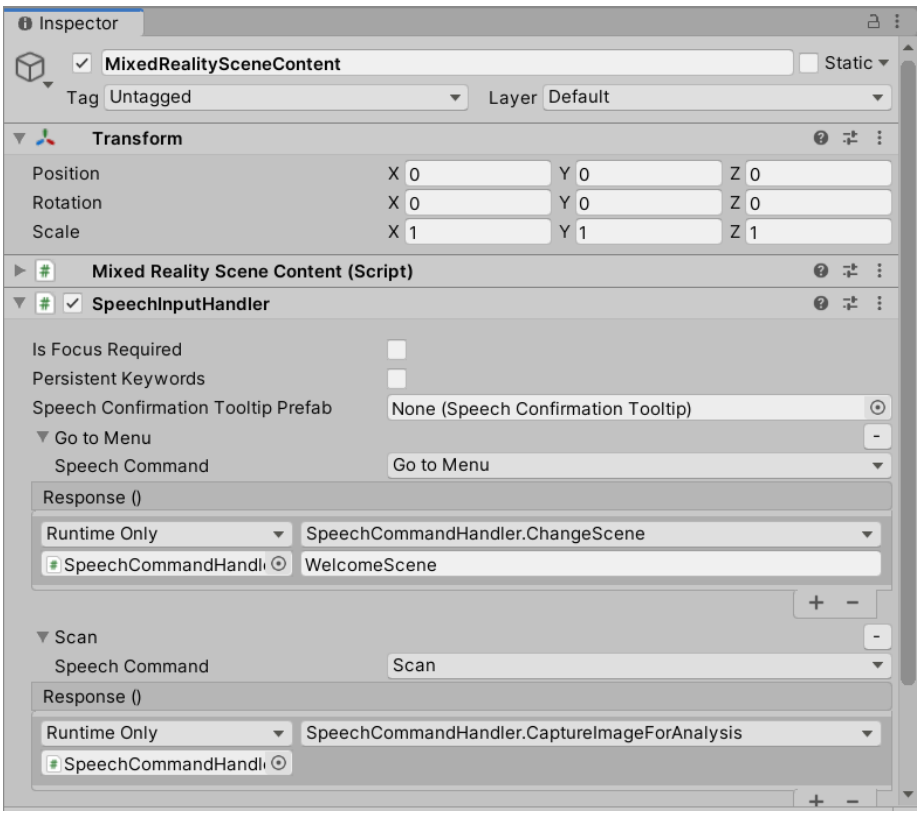


Image 38: Speech input handler for Visual and Audio scan scenes

Regarding the container objects for our object detection holograms, the PredictionObjectsContainer for Visual scan scene is an empty game object, as shown in Image 39, whilst for AudioPredictionObjectsContainer in Audio scan scene, additional configuration is needed to announce the object detection audio description message.

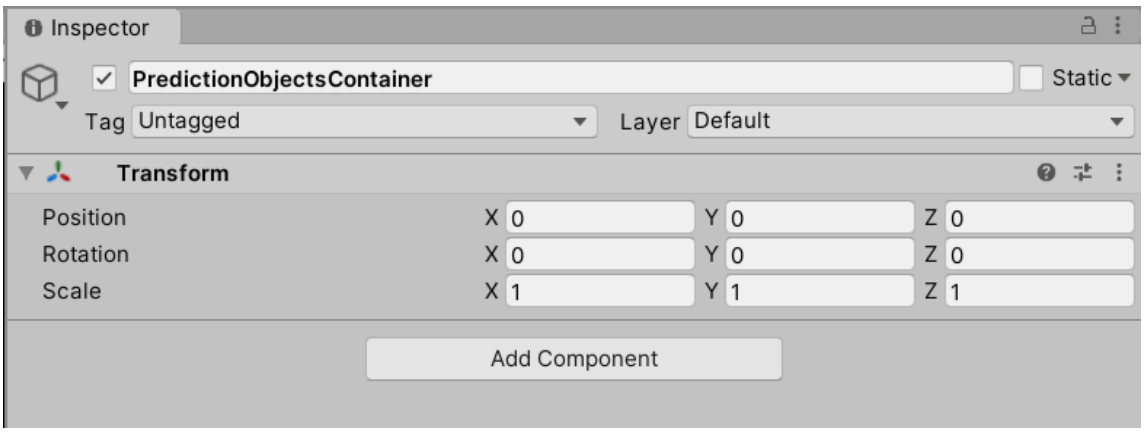


Image 39: Prediction objects container game object for Visual scan scene

Specifically, as depicted in Image 40, we have added the MRTK TextToSpeech script [69]. This class converts a stream into a Unity AudioClip and plays the clip using the AudioSource we supply in the inspector. This allows us to position the voice as desired in 3D space. The Voice variable of this script comes with 4 system voice options to choose from: Default, David, Mark and Zira.

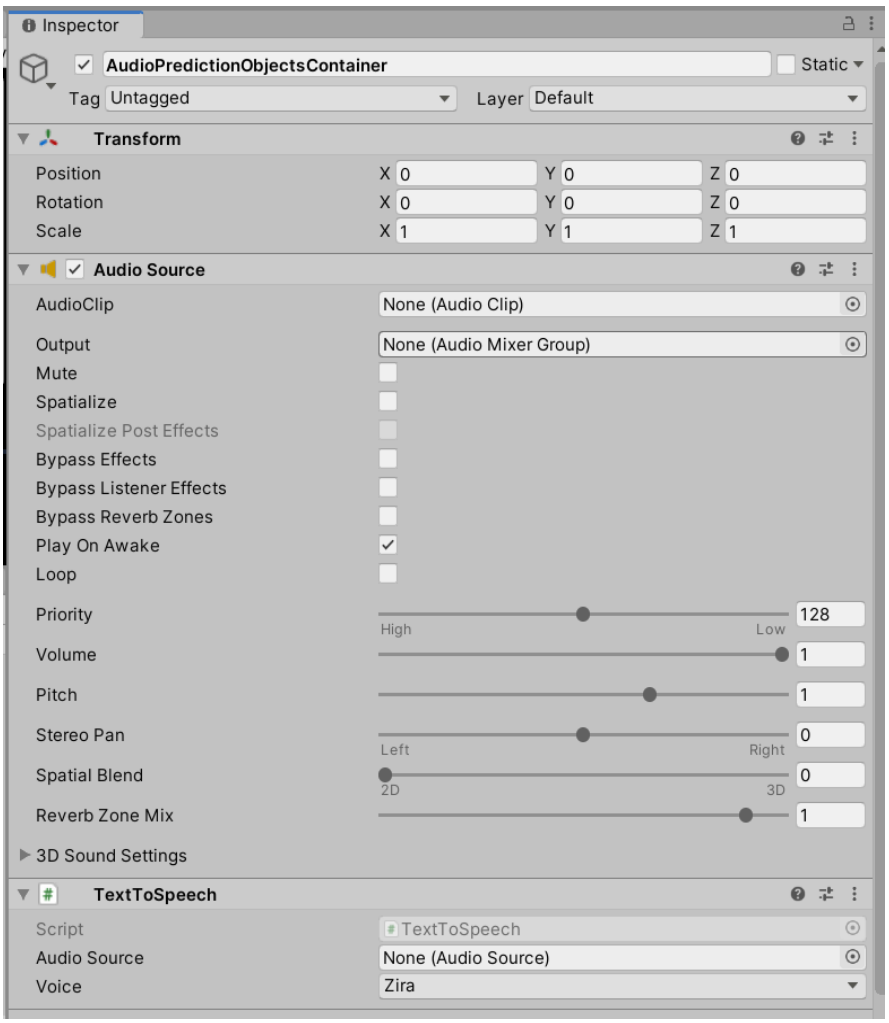


Image 40: TextToSpeech script for Audio scan scene to announce object detection audio message

The coordination of all our defined scripts used to develop each step of the object detection process, is handled from our core script attached to the main camera, which is called ObjectDetectionSceneOrganiser. This script exposes five parameters which we can configurate according to the needs of our scenes. The first one is the asset that we will use for visualizing an object detection result class name, which is the Simple Line Tooltip available from the MRTK foundation package [70]. The second one is the asset that we will use for visualizing an object detection result bounding box, which is a custom prefab asset named BoundingBox, that we will describe in detail in the next section. The third one is used to assign the container game object of the scene to add

dynamically the holograms for each object detection result. As we can see in Image 41, we have set

- The PredictionObjectsContainer for VisualScanScene
- and the AudioPredictionObjectsContainer for AudioScanScene.

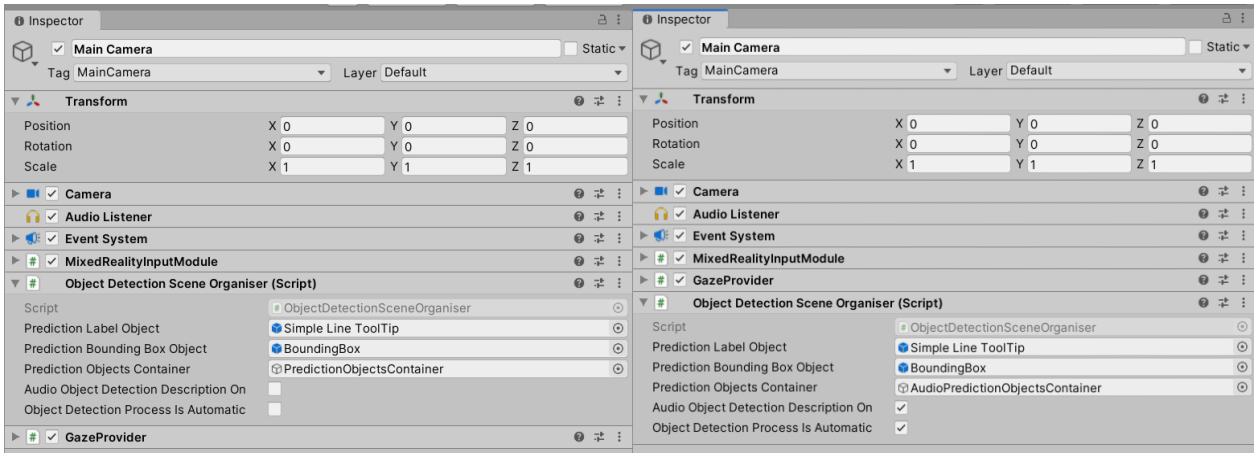


Image 41: Object detection scene organizer script for Visual and Audio scan scenes

Finally, the script exposes two more flag parameters, to configure whether the audio object detection description is enabled or not, and whether the object detection process should start automatically. Both flags are enabled only in Visual scan scene.

6.7 Custom prefab assets for object detection

In this section we will describe in detail the custom prefab assets we have created for object detection scenes.

6.7.1 The Spatial Mapping prefab asset

The SpatialMapping prefab is used to enable and configure actions related to the application's spatial awareness. In this asset we have added the MRTK Spatial Mapping Collider component and our SpatialMapping script. This class will set the Spatial Mapping Collider in the scene so to be able to detect collisions between virtual objects and real objects, a crucial operation for placing the object detection result holograms.

The Unity Editor has low-level Script Reference API for gathering information about surfaces in our project environment. This low-level Script Reference API gives us maximum control over when to query the device for surface changes, and when to create or update the corresponding surface game objects. The Spatial Mapping components allow us to quickly get up and running with mixed reality, without directly

using the low-level Script Reference API. There are two Spatial Mapping components, the Spatial Mapping Renderer and the Spatial Mapping Collider [71].

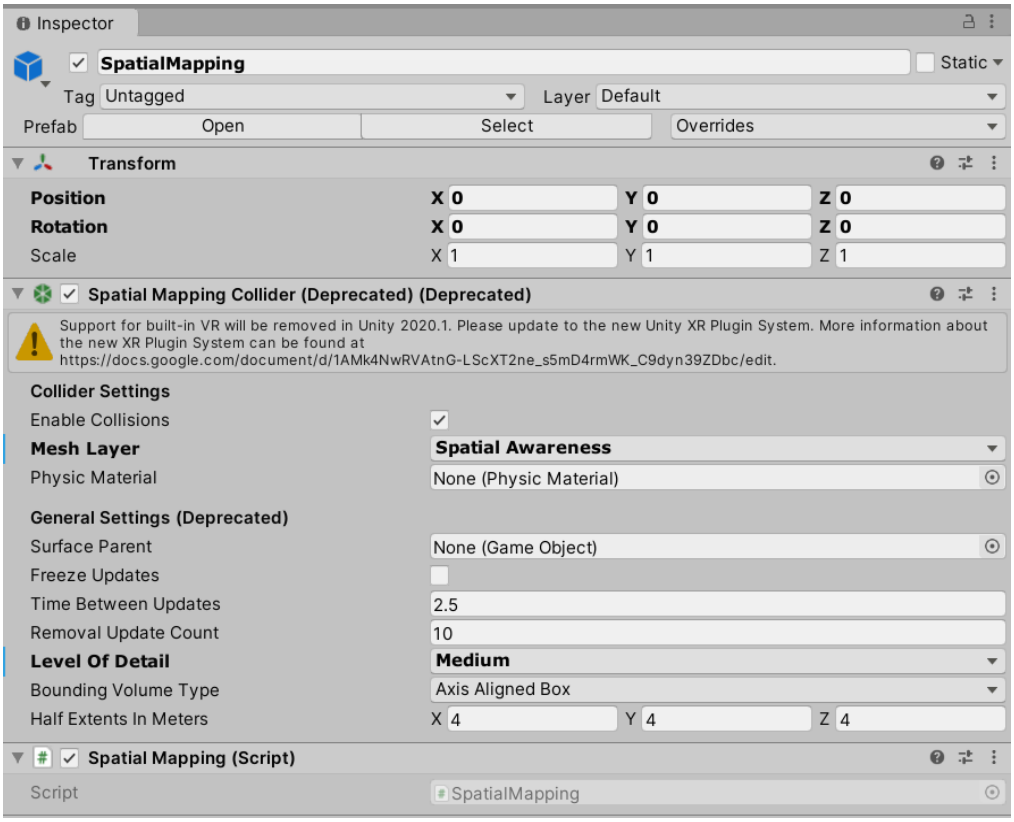


Image 42: The Spatial Mapping prefab asset

The Spatial Mapping Collider component allows holographic content to interact with real-world physical Surfaces. This component handles creating, updating, and destroying the Surface GameObject Colliders in the Scene. The component periodically queries the system for Surface changes in the physical world. When the system reports Surface changes, the Spatial Mapping Collider component prioritizes when each reported Surface is baked by Unity. Every time a Surface is baked by the system, a new GameObject is then generated by Unity, containing a Mesh Filter and Mesh Collider component. A Surface with a Mesh Collider allows raycasts to collide with it as with any other mesh in Unity. We should note that this component only updates the Mesh Collider components of Surface GameObjects, and not the Mesh Renderers, resulting to update with less latency and run faster than Spatial Mapping Mesh Renderers [72].

Let us now describe in detail the configuration parameters for the Spatial Mapping Collider as seen in Image 42:

- Enable Collisions: We have enabled the collisions checkbox, to enable surface mesh colliders. This means that holographic content can collide with surfaces.

- Mesh Layer: This parameter is used for setting the layer property on all the Surface Mesh Colliders. We need to set layers for raycasts. When performing a raycast, we must indicate which Layers we want the ray intersection to test against. By default, Unity assigns all GameObjects to the Default layer. However, it is good practice to assign our GameObjects to a specific layer. Raycasting is an expensive calculation to perform, in that it can slow performance. By using layers, we can filter which GameObjects we are doing our raycast calculations against, and so optimize performance. Therefore, we have set the mesh layer to be equal to the layer we are interested in for performing raycasts to place our object detection holograms, the Spatial awareness layer, to reduce the complexity of raycast tests when doing collisions.
- Surface Parent: With this parameter we can select the Surface Parent GameObject that you want Surface GameObjects generated by Spatial Mapping components to inherit from. We leave this as None to automatically generate a Surface Parent GameObject.
- Freeze Updates: We can enable this parameter to stop the component querying the system for Surface changes. Since the environment for our application is not considered as static, we have not enabled this parameter and leave the component to periodically query the Spatial Mapping data for Surface changes in physical space, which is the default operation.
- Time Between Updates: This parameter defines the time in decimal format seconds between queries for Surface changes in physical space. The default value that we use is 2.5 seconds. We should note that the more regular the queries, the higher the cost in memory, performance, and power.
- Removal Update Count: This parameter defines the number of updates before a Surface GameObject is removed by the system. We can think of an update as a frame in this case. The default value that we use is 10 updates. The removal update countdown begins when Spatial Mapping notifies the component that a Surface GameObject is no longer in the SurfaceObserver’s bounding volume (in that it is no longer within the defined area that the system reports on). This setting allows us to specify the number of updates that should happen after this event before Spatial Mapping removes the Surface GameObject.

- Level of Detail: This parameter defines the quality of the Mesh that the component generates (Low, Medium, or High). The default quality that we use is Medium. The higher the quality, the more refined and accurate the generated Collider or rendered Mesh. Using lower quality settings results in a lower cost in performance and power consumption. We have selected the Medium level of detail as an optimal level that combines an accurate collider for our raycasts to place our holograms in world space and as much as lower cost possible in terms of performance and power usage. In Image 43 below, we can an example of the three level of detail modes for Spatial mapping meshes.

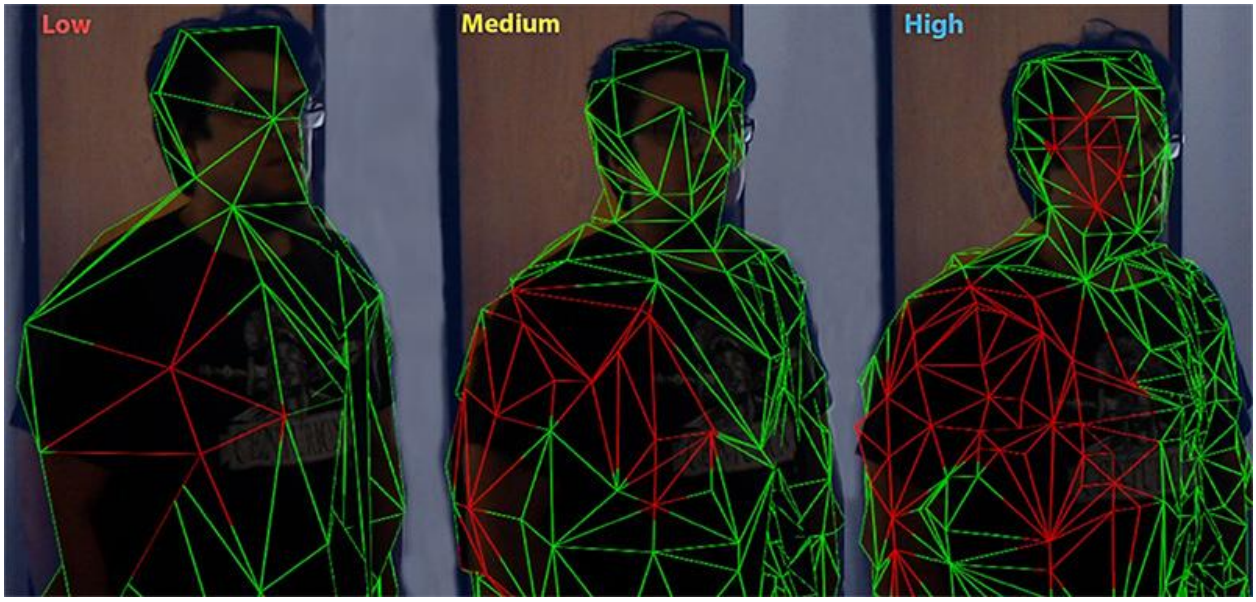


Image 43: The three Level of Detail modes for Spatial Mapping meshes [72]

In our Spatial Mapping script, we set the Spatial Mapping Collider in the scene so to be able to detect collisions between virtual objects and real objects. The PhysicsRaycastMask is used by our GazeCursor as a property with the Raycast call.

```
// This class will set the Spatial Mapping Collider in the scene so to be able to detect collisions between virtual objects and real objects.
Unity Script (1 asset reference) | 3 references
public class SpatialMapping : MonoBehaviour
{

    // Allows this class to behave like a singleton
    public static SpatialMapping Instance;


    // Used by the GazeCursor as a property with the Raycast call
    internal static int PhysicsRaycastMask;


    // The layer to use for spatial mapping collisions
    internal int physicsLayer = 31;


    // Creates environment colliders to work with physics
    [System.Obsolete]
    private SpatialMappingCollider spatialMappingCollider;


    // Initializes this class
    Unity Message | 0 references
    private void Awake()
    {
        // Allows this instance to behave like a singleton
        Instance = this;
    }
```

Image 44: Spatial Mapping script properties

In Start function of the script, which runs at initialization right after Awake method, we initialize and configure the collider, define the physics raycast mask and set the Spatial Mapping GameObject as active.

```
// Runs at initialization right after Awake method
[System.Obsolete]
Unity Message | 0 references
void Start()
{
    // Initialize and configure the collider
    spatialMappingCollider = gameObject.GetComponent<SpatialMappingCollider>();
    spatialMappingCollider.surfaceParent = gameObject;
    spatialMappingCollider.freezeUpdates = false;
    spatialMappingCollider.layer = physicsLayer;

    // define the mask
    PhysicsRaycastMask = 1 << physicsLayer;

    // set the object as active one
    gameObject.SetActive(true);
}
```

Image 45: Spatial mapping script initialization

6.7.2 The Gaze cursor prefab asset

The GazeCursor prefab, is the cursor used in our application for object detection scenes, which follows the user's movement and visualizes the object detection state with its color. When the cursor is green the user can initiate a new object detection scan, whilst when it is red, the user has to wait for the scan to be completed. The gaze cursor works together with the Spatial Mapping prefab to be able to be placed in the scene on top of physical objects.

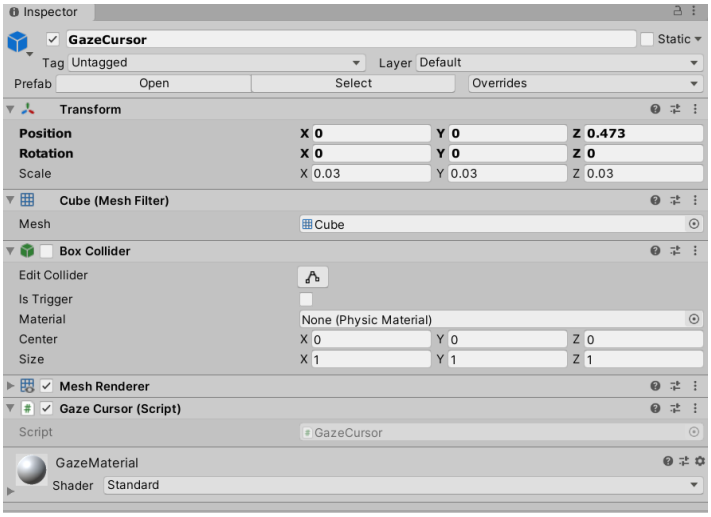


Image 46: The Gaze cursor prefab in Unity inspector

As seen in Image 46, the GazeCursor asset is essentially a Unity primitive cube asset, with a special material. The GazeCursor script attached is responsible for setting up the cursor in the correct location in real space, by making use of the Spatial Mapping Collider. During the initialization, we access the mesh renderer component and save it as a reference in our script. The core ObjectDetectionSceneOrganiser script for object detection keeps a reference in our gaze cursor for changing the cursor color based on the object detection step. We set this reference along with setting the size of the cursor.

```
// The cursor (this object) mesh renderer
private MeshRenderer meshRenderer;

// Runs at initialization right after the Awake method
Unity Message | 0 references
void Start()
{
    // Grab the mesh renderer that is on the same object as this script.
    meshRenderer = gameObject.GetComponent<MeshRenderer>();

    // Set the gaze cursor reference in scene organiser
    ObjectDetectionSceneOrganiser.Instance.gazeCursor = gameObject;
    gameObject.GetComponent<Renderer>().material.color = Color.green;

    // Change the size of the cursor
    gameObject.transform.localScale = new Vector3(0.01f, 0.01f, 0.01f);
}
```

Image 47: Gaze cursor script initialization

Once per frame, when the update function of the gaze cursor is called, we need to update the position of our gaze cursor to follow the user's gaze, as described in [73]. From the camera game object, we obtain the user's head position and gaze direction. We then perform a raycast operation into the world and move the cursor to the point the

```
// Update is called once per frame
Unity Message | 0 references
void Update()
{
    // Do a raycast into the world based on the user's head position and orientation.
    Vector3 headPosition = Camera.main.transform.position;
    Vector3 gazeDirection = Camera.main.transform.forward;

    if (Physics.Raycast(headPosition, gazeDirection, out RaycastHit gazeHitInfo, 30.0f, SpatialMapping.PhysicsRaycastMask))
    {
        // If the raycast hit a hologram, display the cursor mesh.
        meshRenderer.enabled = true;
        // Move the cursor to the point where the raycast hit.
        // Rotate the cursor to hug the surface of the hologram.
        transform.SetPositionAndRotation(gazeHitInfo.point, Quaternion.FromToRotation(Vector3.up, gazeHitInfo.normal));
    }
    else
    {
        // If the raycast did not hit a hologram, hide the cursor mesh.
        meshRenderer.enabled = false;
    }
}
```

Image 48: Gaze cursor position update to match the user's gaze

raycast did hit. We also rotate the cursor to hug the surface of the hologram. In Image 49, we can see our application's gaze cursor for object detection in green and red state.

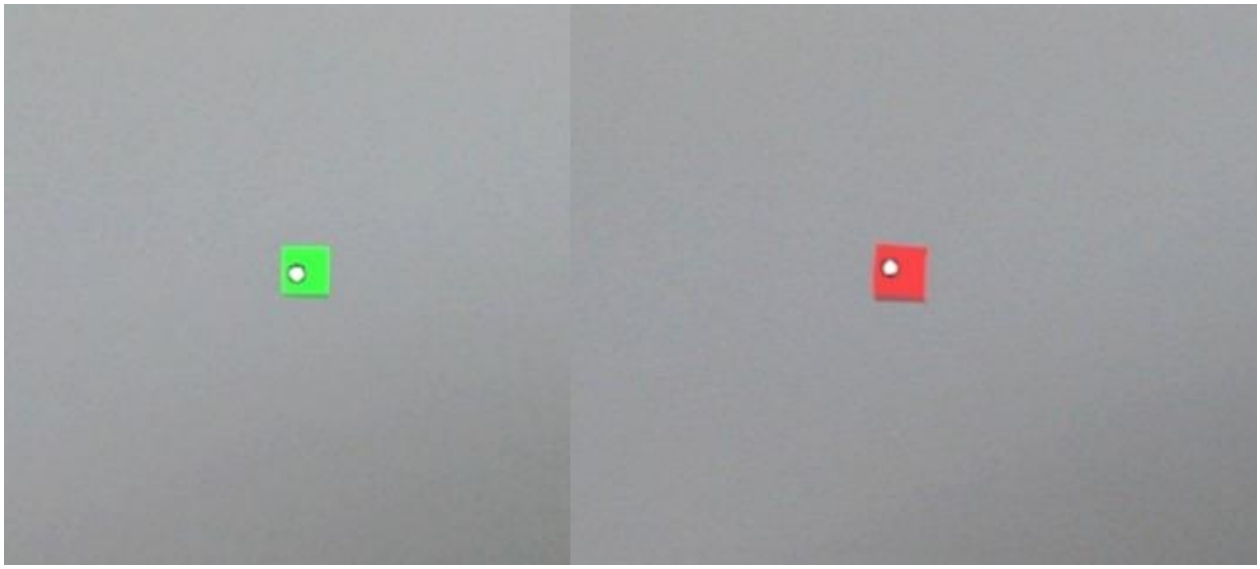


Image 49: Our application's gaze cursor in green and red state

6.7.3 The Object detection result bounding box prefab asset

To visualize our object detection results, we need apart from a label hologram for the result's class for which we use the MRTK Tooltip prefab, a hologram that serves as a bounding box for each result. The data we have available after applying the necessary data transformations upon the object detection server response, are the 3D world coordinates of the four vertices that construct an object detection result bounding box. To create our bounding box hologram, we need to draw a rectangle that encloses these four vertices world points. In this case we can use Unity's LineRenderer class [74].

The Line Renderer component takes an array of two or more points in 3D space and draws a straight line between each one. We can use a Line Renderer to draw anything from a simple straight line to a complex spiral. The line is always continuous. Furthermore, the Line Renderer does not render lines that have a width in pixels. It renders polygons that have a width in world units.

As we can see in Image 50, for creating our Bounding box prefab asset, we have added a Line Renderer component, using the default line shader. We have also enabled the "Loop" option, to obtain a closed shape. We have also attached the BoundingBox script, which help up to initialize an object detection bounding box, by setting the four corners in 3D space, after getting the result response from our server.

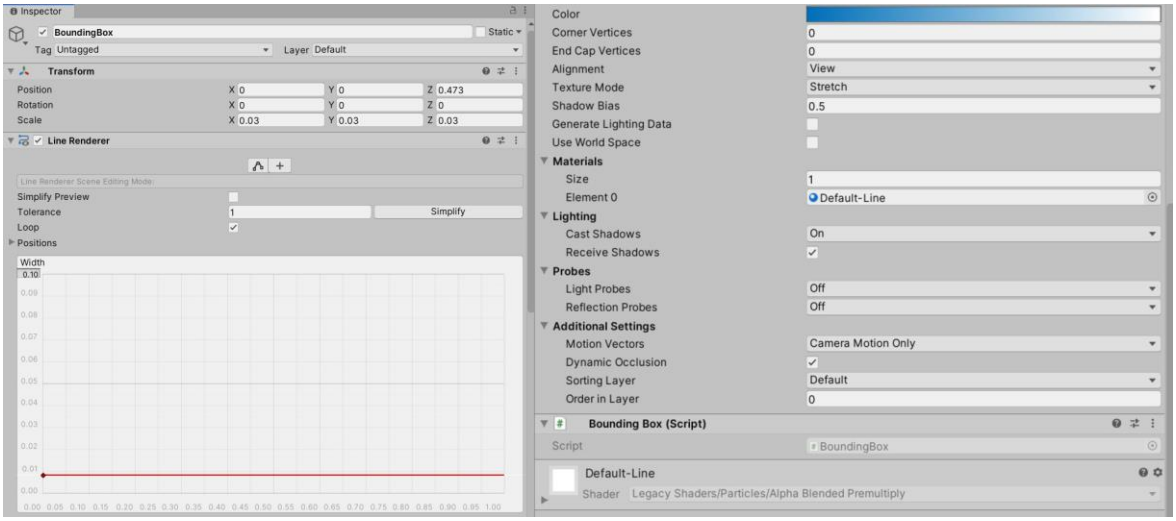


Image 50: The Bounding Box prefab is Unity inspector

The initialize function of the BoundingBox script is called from our core ObjectDetectionSceneOrganiser script when rendering the object detection result holograms. It takes the number of points to add to the Line Renderer and an array with the 3D world points to add as parameters. We access the Line Renderer component attached to the game object and pick a random, saturated color for the line. An important setting is to set the Line Renderer property “userWorldSpace” to true, as we have to connect 3D world points and the lines should be rendered around world origin. We set the “loop” property to true to obtain a closed shape. Finally, we set the points to connect by setting the positions count and the positions array. We set the points to connect as an array instead of adding the points one by one to the Line Renderer, because it is more efficient in terms of rendering.

```
// Script for initializing an object detection bounding box, by setting the four corners in 3D space
[RequireComponent(typeof(LineRenderer))]
Unity Script (1 asset reference) | 1 reference
public class BoundingBox : MonoBehaviour
{
    1 reference
    public void Initialize(int positionsCount, Vector3[] pointsArray)
    {
        // Get a line renderer to connect the four corners
        LineRenderer lineRenderer = gameObject.GetComponent<LineRenderer>();

        // Pick a random, saturated and not-too-dark color for the line
        SetLineSingleColor(lineRenderer, Random.ColorHSV(0f, 1f, 1f, 1f, 0.5f, 1f));

        // User world space system for positions
        lineRenderer.useWorldSpace = true;

        lineRenderer.loop = true;
        lineRenderer.positionCount = positionsCount;
        lineRenderer.SetPositions(pointsArray);


    }

    1 reference
    private void SetLineSingleColor(LineRenderer lineRendererToColor, Color color)
    {
        lineRendererToColor.startColor = color;
        lineRendererToColor.endColor = color;
    }
}
```

Image 51: The Object detections result Bounding Box script

In Image 52, we can see the bounding box hologram along with the MRTK tooltip, that is rendered to enclose the book detected from our application.

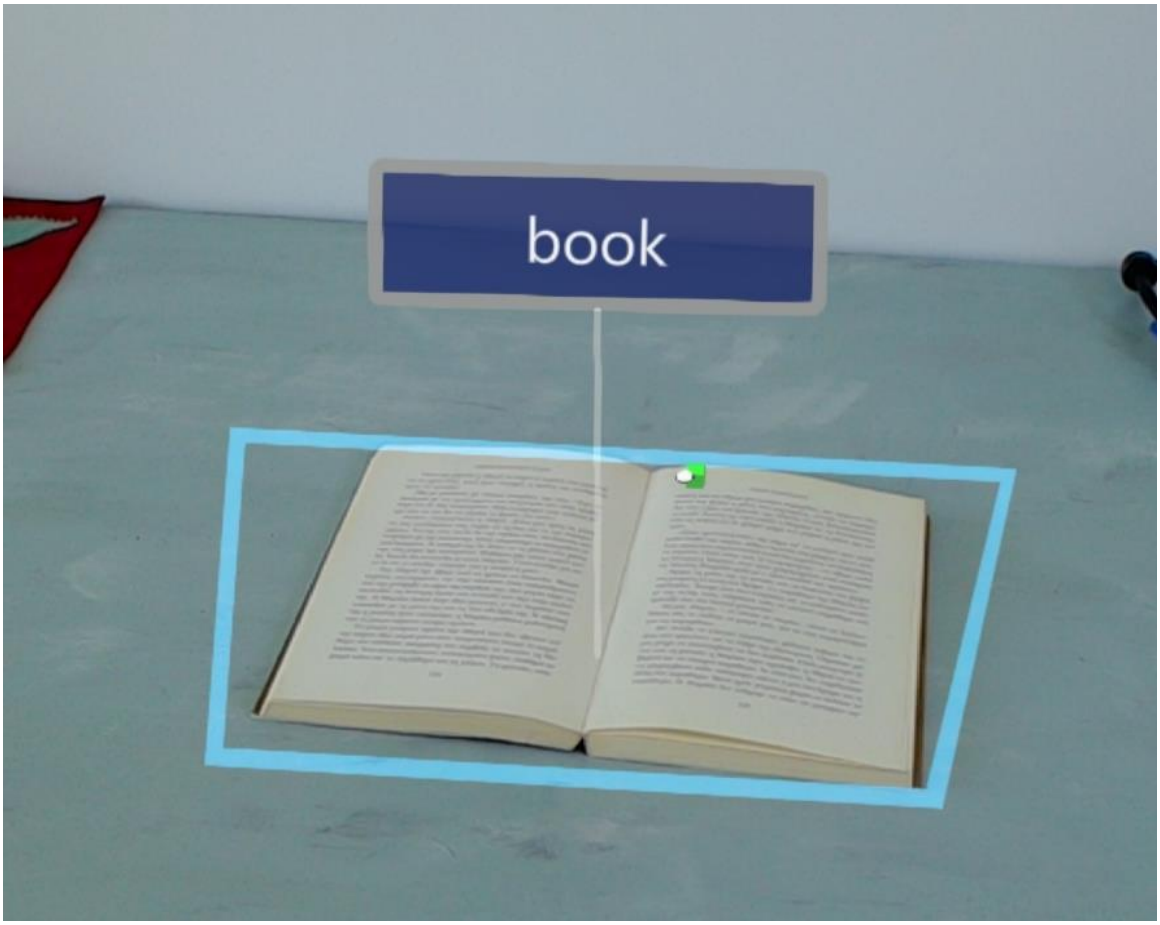


Image 52: The Bounding Box hologram that enclosed a book detected from our application

6.7.4 The Audio effects player prefab asset

To enhance the user's experience and provide audio feedback for the operations that take place during the object detection process, we have set some sound effects. Sound effects take place when we perform a camera capture for object detection, and when we retrieve the object detection result from our server. There is a different sound effect for success (where objects are detected in the user's view) and failure (when no object are detected or there was an error during server communication).

For this purpose, we have created a prefab asset, the AudioEffectsPlayer prefab. In order to play back sounds in a 3D environment, we have to attach an Audio source component to our game object [75]. As depicted in Image 53, the Audio source is added with the default parameters kept for its configuration. For playing a certain sound effect on demand, during the object detection part that we need, we use the Audio Effects

Player Helper script attached to the game object, which exposes as a parameter a list of audio clips that we can play.

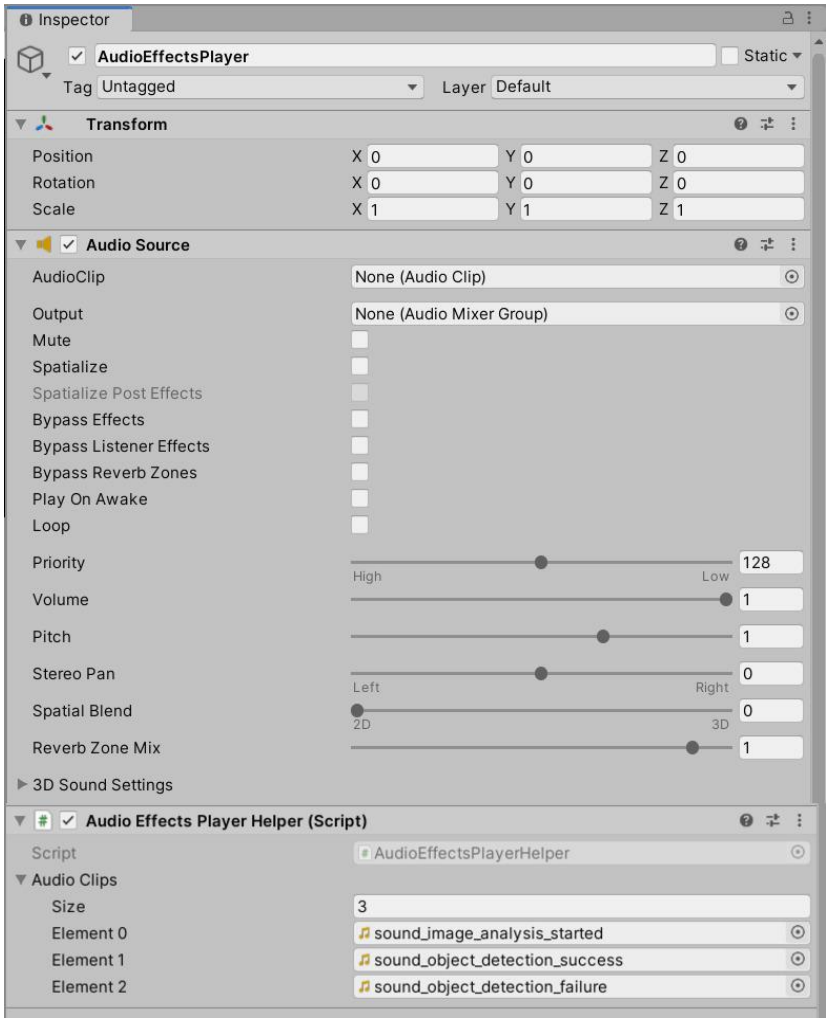


Image 53: The Audio effects player prefab in Unity inspector

An AudioClip stores the audio file either compressed as ogg vorbis or uncompressed. AudioClips are referenced and used by AudioSources to play sounds [76].

The sound effects that we set as the parameter values, are stored as .mp3 files in our project's resources folder. In our script, during initialization, we set the Audio effects player helper reference object we keep to the core Object Detection Scene organizer script, for calling the PlayAudio function with a certain audio clip, as seen in Image 54. The audio source component property is assigned with the Audio source we have attached to our game object.

During the object detection parts that we need to play a certain audio effect, we call the PlayAudio function, passing as a parameter the audio clip index from the list we have set during the configuration of the prefab. If the index is valid, we set the audio clip from

the list with this index as the clip of the audio source and call the Play function to play this certain audio clip.

```
//This script allows us to toggle music to play and stop.
//Assign an AudioSource to the GameObject.
Unity Script (1 asset reference) | 1 reference
public class AudioEffectsPlayerHelper : MonoBehaviour
{
    private AudioSource audioSource;

    // A list with audio clips we wish to play
    public List<AudioClip> audioClips;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        // Set the audio effects player helper reference in scene organiser
        ObjectDetectionSceneOrganiser.Instance.audioEffectsPlayerHelper = gameObject;

        audioSource = GetComponent<AudioSource>();
    }

    1 reference
    public void PlayAudio(int audioClipIndex)
    {
        if (audioSource != null && audioClips != null && audioClipIndex >= 0 && audioClipIndex < audioClips.Count)
        {
            audioSource.clip = audioClips[audioClipIndex];
            audioSource.Play();
        }
    }
}
```

Image 54: The Audio effects player helper script

6.8 The Object detection steps through our scripts

In this section we will describe in detail the scripts developed for each step of the object detection process in our application.

6.8.1 The Image capture script

The Image Capture class is responsible for:

- Handling Tap gestures from the user
- Starting the periodic operation of environment scanning when it comes to Audio scan
- Capturing an image using the HoloLens camera and sending it along with camera resolution and a copy of the Camera's transform for analysis.

During initialization, the Start method is called. We query for the flag parameter "objectDetectionProcessIsAutomatic" held by our core ObjectDetectionSceneOrganiser script, to check if the object detection process should start automatically, which in our

application applies only for Audio scan scene. In this case, we start a coroutine for periodic checks related to the camera frames similarity, with a purpose to apply the object detection operation only in cases where the user's view has changed to a significant extent and limit complex computations when the user's view has no important differences considering that have already performed a scan for these environment circumstances in the past.

For Visual scan scene, the detection process does not start automatically, but based on the user's demand with a tap gesture or a voice command. The "Scan" voice command is handled from the Speech Input Handler that is added on the scene's MixedRealitySceneContent. In order to respond to the user's tap gesture, we have to subscribe to the Microsoft HoloLens API gesture recognizer, set as the recognizable gestures the GestureSettings.Tap, define the method to call when a tap gesture is recognized and register the API listener with "StartCapturingGestures" function.

In both cases of Visual and Audio scan during initialization, we set the camera resolution that we are going to use for our camera captures, to be the highest possible that the device supports, which equals to 720p (1268x720) [77].

The function that we call either on a tap gesture, voice command, or periodically with a period defined by the class property "automaticImageCapturePeriod", which equals to 30 seconds is called "StartImageCaptureProcess".

```
void Start()
{
    // Subscribing to the Microsoft HoloLens API gesture recognizer to track user tap gestures
    // Only if the object detection process does not start automatically
    if (!ObjectDetectionSceneOrganiser.Instance.objectDetectionProcessIsAutomatic)
    {
        recognizer = new GestureRecognizer();
        recognizer.SetRecognizableGestures(GestureSettings.Tap);
        recognizer.Tapped += TapHandler;
        recognizer.StartCapturingGestures();
    }
    else
    {
        StartCoroutineForPeriodicFrameSimilarityChecks();
    }

    // Set the camera resolution to be the highest possible
    cameraResolution = PhotoCapture.SupportedResolutions.OrderByDescending
        ((res) => res.width * res.height).First();
}


// Respond to Tap Input.
1 reference
private void TapHandler(TappedEventArgs obj) => StartImageCaptureProcess();

1 reference
private void StartCoroutineForPeriodicFrameSimilarityChecks()
{
    StartCoroutine(nameof(PeriodicFrameSimilarityChecks));
}

1 reference
private IEnumerator PeriodicFrameSimilarityChecks()
{
    while (true)
    {
        // Start the image capture process without a player action and wait for the period time
        StartImageCaptureProcess();
        yield return automaticImageCapturePeriod;
    }
}
```

Image 55: Initialization actions in Image Capture script

In StartImageCaptureProcess function, we firstly check our script property "captureIsActive", which defines if a scanning operation is in progress or not. In case that no scanning tasks run in the background, we set this flag to true to lock the capture task and prevent new scans to take place before we get a respond and render the results from the scan that is ongoing. We also set the gaze cursor color to red using the reference held by the ObjectDetectionSceneOrganiser script instance, to provide visual feedback to the user that the device is currently busy with an object detection scan.

Subsequently, we invoke the "ExecuteImageCaptureAndAnalysis" function to begin the capture process. Specifically, using the PhotoCapture API [78], using the "PhotoCapture.CreateAsync" function we create an instance of a PhotoCapture object that can be used to capture photos from the device. We then set the camera parameters for our capture operation. Since we do not wish to include any holograms present in the user's view to our capture, we set the hologram opacity to zero. Additionally, we set the camera resolution width and height to be the highest possible that the device supports using our “cameraResolution” property initialized previously in Start function. Finally, we request the output pixel format of the camera capture object to be JPEG. We activate the camera by calling the "StartPhotoModeAsync" function on the capture object acquired and take a picture asynchronously using “TakePhotoAsync” call. When this API function returns, we set to call our defined “OnCapturedPhotoToMemory” function.

```
3 references
public void StartImageCaptureProcess()
{
    // When a tap gesture is identified, we respond to it only if the camera capture is not active
    if (!captureIsActive)
    {
        captureIsActive = true;

        // Set the cursor color to red, to denote that the camera is busy.
        // When the cursor is green, it means the camera is available to take the image.
        if (ObjectDetectionSceneOrganiser.Instance.gazeCursor != null)
        {
            ObjectDetectionSceneOrganiser.Instance.gazeCursor.GetComponent<Renderer>().material.color = Color.red;
        }

        // Begin the capture process
        Invoke(nameof(ExecuteImageCaptureAndAnalysis), 0);
    }
}

// Begin process of image capturing and send to our Object Detection Service for analysis.
1 reference
private void ExecuteImageCaptureAndAnalysis()
{
    // Begin capture process, set the image format (without showing holograms)
    PhotoCapture.CreateAsync(false, delegate (PhotoCapture captureObject)
    {
        photoCaptureObject = captureObject;

        CameraParameters cameraParameters = new CameraParameters
        {
            hologramOpacity = 0.0f,
            cameraResolutionWidth = cameraResolution.width,
            cameraResolutionHeight = cameraResolution.height,
            pixelFormat = CapturePixelFormat.JPEG
        };

        // Activate the camera
        photoCaptureObject.StartPhotoModeAsync(cameraParameters, delegate (PhotoCapture.PhotoCaptureResult result)
        {
            // Take a picture
            photoCaptureObject.TakePhotoAsync(OnCapturedPhotoToMemory);
        });
    });
}
```

Image 56: Starting the image capture process in Image Capture script

When the camera capture is completed, the result of the capture that we are interested in is the photo capture frame. Using the photo capture frame object, we can have access to the frame native image data and spatial maps which will be necessary for transforming a certain 2D pixel point from the camera capture image to a 3D world point to place our object detection results holograms.

Regarding the spatial maps, we try to get the camera to world and projection matrix, using "TryGetCameraToWorldMatrix" and "TryGetProjectionMatrix" API function respectively. These methods will return the spatial maps at the matrices we pass as parameters at the time the photo was captured if location data if available. In this case, since our spatial matrices are valid, the function return a true Boolean value. If location data is unavailable then the matrices be set to the identity matrix, and the functions will return false as a result. Using the "CopyRawImageDataIntoBuffer" API function on the photo capture frame object, we also retrieve the raw media buffer data in JPEG pixel format into a byte list. If all the aforementioned data are valid, we construct a domain model with name "ImageCaptureResultData", which includes the result data we get from this script, the native data and spatial maps of the camera capture, along with the camera resolution info that we used for the capture.

Finally, we call the "StopPhotoModeAsync" API call on the photo capture object to asynchronously stop the photo mode and deactivate the camera. After this deactivation is completed, our defined "OnStoppedPhotoMode" function is set to be called. We first dispose the photo capture object from memory and then we start the process for camera frames similarity check or object detection as defined in CameraFramesSimilarityCheck and YoloObjectDetectionAnalyser scripts respectively. We will describe these scripts in detail in the following sections. In both cases we pass as parameters the imageCaptureResultData domain model we created previously, holding all necessary results from this script. The functions for getting the result data from the camera capture frame are described in Image 57.

The last function we define in this script is the “ResetImageCapture” function. We call it when the scanning process is finished, or in case one of the result data we need to proceed to the next step of our scanning process is not valid, either one of the spatial maps or the native image data. In this function, as we can see in Image 58, we reset our flag “captureIsActive” to false for new scans to start, reset the gaze cursor color to green and stop the capture loop in case it is active.

```
// Register the full execution of the Photo Capture.
1 reference
void OnCapturedPhotoToMemory(PhotoCapture.PhotoCaptureResult result, PhotoCaptureFrame photoCaptureFrame)
{
    // Take camera to world and projection matrix
    bool isCameraToWorldMatrixValid = photoCaptureFrame.TryGetCameraToWorldMatrix(out Matrix4x4 cameraToWorldMatrix);
    bool isProjectionMatrixValid = photoCaptureFrame.TryGetProjectionMatrix(Camera.main.nearClipPlane, Camera.main.farClipPlane,out Matrix4x4 projectionMatrix);
    Debug.Log("isCameraToWorldMatrixValid is " + isCameraToWorldMatrixValid + " and isProjectionMatrixValid is " + isProjectionMatrixValid);

    // Copy raw image data to photoBuffer
    var photoBuffer = new List<byte>();
    if (photoCaptureFrame.pixelFormat == CapturePixelFormat.JPEG)
    {
        photoCaptureFrame.CopyRawImageDataIntoBuffer(photoBuffer);
    }

    if (isCameraToWorldMatrixValid && isProjectionMatrixValid && photoBuffer != null)
    {
        imageCaptureResultData = new ImageCaptureResultData(photoBuffer, cameraResolution, cameraToWorldMatrix, projectionMatrix);
    }

    // Call StopPhotoMode once the image has successfully captured to deactivate our camera
    photoCaptureObject.StopPhotoModeAsync(OnStoppedPhotoMode);
}

// The camera photo mode has stopped after the capture.
// Begin the image analysis process.
1 reference
void OnStoppedPhotoMode(PhotoCapture.PhotoCaptureResult result)
{
    Debug.LogFormat("Stopped Photo Mode");

    // Dispose from the object in memory and request the image analysis
    photoCaptureObject.Dispose();
    photoCaptureObject = null;

    if (imageCaptureResultData != null)
    {
        if (!ObjectDetectionSceneOrganiser.Instance.objectDetectionProcessIsAutomatic)
        {
            // Call the image analysis
            StartCoroutine(YoloObjectDetectionAnalyser.Instance.AnalyseLastImageCaptured(imageCaptureResultData));
        }
        else
        {
            // Call the camera frames similarity check
            StartCoroutine(CameraFramesSimilarityCheck.Instance.CheckPreviousAndCurrentFramesCapturedForSimilarity(imageCaptureResultData));
```

Image 57: Acquiring native image data and spatial maps from photo capture object

```
// Stops all capture pending actions
2 references
internal void ResetImageCapture()
{
    captureIsActive = false;

    // Set the cursor color to green
    ObjectDetectionSceneOrganiser.Instance.gazeCursor.GetComponent<Renderer>().material.color = Color.green;

    // Stop the capture loop if active
    CancelInvoke();
}
```

Image 58: Resetting the image capture operations

6.8.2 The Camera frames similarity check script

In case the user has selected the Audio scan scene, after retrieving the result data object from the Image Capture script as described in the previous section, we are transferred to the CameraFramesSimilarityCheck script, which is responsible for performing a POST request to our development server containing the current and preceding camera frames native data, and based on the server response, if the frames

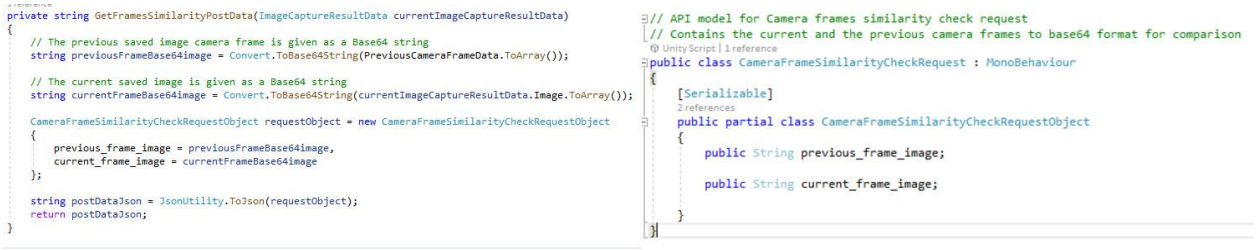
are not annotated as similar, the camera capture result data are transferred to the script dedicated for the object detection web request.

In this script, we keep a property named “PreviousCameraFrameData”, a byte list that holds the native image data bytes in JPEG format as returned from the Image capture script, for the previous camera frame captured. During the initialization Awake method, this property is set to null since we are interested in it only after the second execution of the periodic scanning. The script exposes a public method called “CheckPreviousAndCurrentFramesCapturedForSimilarity“, which is used for similarity checks between the current and the preceding camera frames and takes the image capture result data object as a parameter. In case this is the first periodic execution of the function and our “PreviousCameraFrameData” is empty, we proceed with calling the method for applying object detection on this camera frame, offered by the “YoloObjectDetectionAnalyser” script, described in the next section. We also save this frame’s native data to the “PreviousCameraFrameData” property, for the similarity check of the next periodic execution. At this point, we construct a Unity POST web request to the endpoint for frames similarity check in our development server. The post data sent in this request are constructed in the “GetFramesSimilarityPostData” method.

```
private IList<byte> PreviousCameraFrameData;

// Initializes this class
Unity Message | 0 references
private void Awake()
{
    // Allows this instance to behave like a singleton
    Instance = this;
    // Initialize the previous camera frame data to null
    PreviousCameraFrameData = null;
}

// Call our Camera Frames Similarity check Service to submit the previous and current frames data.
// Obtains the result of the frames similarity with a boolean flag which denotes if frames are similar or not.
1 reference
public IEnumerator CheckPreviousAndCurrentFramesCapturedForSimilarity(ImageCaptureResultData currentImageCaptureResultData)
{
    Debug.Log("Checking camera frames similarity...");

    if (PreviousCameraFrameData == null)
    {
        // In case this similarity check has not been called previously, start the object detection API call,
        // and save the current image frame for the next call of frames similarity
        PreviousCameraFrameData = currentImageCaptureResultData.Image;
        StartCoroutine(YoloObjectDetectionAnalyser.Instance.AnalyseLastImageCaptured(currentImageCaptureResultData));
    }
    else
    {
        using (UnityWebRequest framesSimilarityWebRequest = UnityWebRequest.Post(framesSimilarityEndpoint, string.Empty))
        {
            // The upload handler will help uploading the post data
            byte[] jsonToSend = new System.Text.UTF8Encoding().GetBytes(GetFramesSimilarityPostData(currentImageCaptureResultData));
            framesSimilarityWebRequest.uploadHandler = new UploadHandlerRaw(jsonToSend);

            // The download handler will help receiving the analysis from camera frames similarity API call
            framesSimilarityWebRequest.downloadHandler = new DownloadHandlerBuffer();

            framesSimilarityWebRequest.SetRequestHeader("Content-Type", "application/json");

            // Send the request and wait for the reponse
            yield return framesSimilarityWebRequest.SendWebRequest();
```

Image 59: Initializations and server request execution in Camera frames similarity check script

In this method, as we can see in Image 60, the current and preceding camera frame native image byte data are converted to Base64 strings. Base64 is an encoding algorithm that converts any characters, binary data, and even images or sound files into a readable string, which can be saved or transported over the network without data loss. The characters generated from Base64 encoding consist of Latin letters, digits, plus, and slash. Base64 images are primarily used to embed image data within other formats like JSON used in our case. In order to transfer our data through our RESTful service, we have created an API model for our web request, called “CameraFrameSimilarityCheckRequest”, which contains the current and preceding camera frames image data as Base64 strings. Using the Unity’s JsonUtility.ToJson utility function, we generate a JSON representation of the public fields of our API request model. The object's data in JSON string format are finally converted to bytes to be given as an input in the request upload handler.



```
private string GetFramesSimilarityPostData(ImageCaptureResultData currentImageCaptureResultData)
{
    // The previous saved image camera frame is given as a Base64 string
    string previousFrameBase64image = Convert.ToBase64String(PreviousCameraFrameData.ToArray());

    // The current saved image is given as a Base64 string
    string currentFrameBase64image = Convert.ToBase64String(currentImageCaptureResultData.Image.ToArray());

    CameraFrameSimilarityCheckRequestObject requestObject = new CameraFrameSimilarityCheckRequestObject
    {
        previous_frame_image = previousFrameBase64image,
        current_frame_image = currentFrameBase64image
    };

    string postDataJson = JsonUtility.ToJson(requestObject);
    return postDataJson;
}
```

```
// API model for Camera frames similarity check request
// Contains the current and the previous camera frames to base64 format for comparison
Unity Script | 1 reference
public class CameraFrameSimilarityCheckRequest : MonoBehaviour
{
    [Serializable]
    2 references
    public partial class CameraFrameSimilarityCheckRequestObject
    {
        public String previous_frame_image;

        public String current_frame_image;

    }
}
```

Image 60: Constructing the web request POST data for camera frames similarity check

When we receive the response from server, in case of a valid response, as depicted in Image 61, we first have to deserialize the JSON representation into our API response model called "CameraFramesSimilarityAPIResult", which contains a Boolean flag denoting if the camera frames submitted are similar or not. If the frames are not similar, we proceed with applying object detection on this camera frame, otherwise we reset the object detection process. In both cases, we save the current frame's native image data for the next periodic execution. The object detection process is also reset in case of a network error during the communication with the server, or in case we receive an invalid response.

```
if (framesSimilarityWebRequest.isNetworkError || framesSimilarityWebRequest.isHttpError)
{
    Debug.Log("Communication with server for frames similarity : Network error");
    Debug.Log("Error is " + framesSimilarityWebRequest.error);
    ObjectDetectionSceneOrganiser.Instance.ResetDetectionProcessOnError();
}
else
{
    string jsonResponse = framesSimilarityWebRequest.downloadHandler.text;

    Debug.Log("frames similarity response: " + jsonResponse);

    // The response is in JSON format, therefore it needs to be deserialized in our API model
    CameraFramesSimilarityAPIResult similarityCheckResult = new CameraFramesSimilarityAPIResult();
    similarityCheckResult = CameraFramesSimilarityAPIResult.CreateFromJSON(jsonResponse);

    if (similarityCheckResult != null)
    {
        // Save the current image frame for the next call of frames similarity
        PreviousCameraFrameData = currentImageCaptureResultData.Image;
        // The frames are not similar, hence start the object detection API call for the next frame
        if (!similarityCheckResult.areFramesSimilar)
        {
            StartCoroutine(YoloObjectDetectionAnalyser.Instance.AnalyseLastImageCaptured(currentImageCaptureResultData));
        }
        else
        {
            ObjectDetectionSceneOrganiser.Instance.ResetDetectionProcess();
        }
    }
    else
    {
        Debug.Log("Similarity check response is null");
        // Play sound effect for object detection failure
        ObjectDetectionSceneOrganiser.Instance.ResetDetectionProcessOnError();
    }
}
```

Image 61: Handling the server response for camera frames similarity check

6.8.3 The YOLO Object Detection Analyzer script

After retrieving the result from Image Capture script for Visual scan, or when the current and preceding camera frames are not considered similar for Audio scan, we proceed with calling “AnalyseLastImageCaptured” public function to submit the camera frame captured for object detection analysis. The function receives the image capture result data as an input parameter.

```
// Call our YOLO Object detection Service to submit the image.
// Obtains the results of the analysis of the image, captured by the ImageCapture class.
3 references
public IEnumerator AnalyseLastImageCaptured(ImageCaptureResultData imageCaptureResultData)
{
    Debug.Log("Analyzing...");

    // Play sound effect for image analysis process start
    ObjectDetectionSceneOrganiser.Instance.PlaySoundForImageAnalysisStart();

    using (UnityWebRequest analysisWebRequest = UnityWebRequest.Post(predictionEndpoint, string.Empty))
    {
        // The upload handler will help uploading the post data
        byte[] jsonToSend = new System.Text.UTF8Encoding().GetBytes(GetImageAnalysisPostData(imageCaptureResultData));
        analysisWebRequest.uploadHandler = new UploadHandlerRaw(jsonToSend);

        // The download handler will help receiving the analysis from YOLO server
        analysisWebRequest.downloadHandler = new DownloadHandlerBuffer();

        analysisWebRequest.SetRequestHeader("Content-Type", "application/json");

        // Send the request and wait for the reponse
        yield return analysisWebRequest.SendWebRequest();
```

Image 62: The server request for Object detection analysis on a camera frame

We first play a sound effect for image analysis start, as audio feedback to the user for the analysis background task. We then construct a Unity POST web request to the endpoint for image analysis in our development server. The post data sent in this request are constructed in the “GetImageAnalysisPostData” method.

In this method, as we can see in Image 63, the camera frame native image byte data are converted to Base64 string. We also add a flag to denote if the audio description feature is enabled or not, based on the configuration of the Object Detection Scene Organizer script. To transfer our data through our RESTful service, we have created an API model for our web request, called “ImageAnalysisRequestObject”, which contains the camera frame image data as Base64 string and a Boolean flag to denote if we should receive a descriptive text for the objects detected in the scene as audio feedback of the analysis, which applies for Audio scan. Using the Unity’s JsonUtility.ToJson utility function, we generate a JSON representation of the public fields of this API request model. The object's data in JSON string format are finally converted to bytes to be given as an input in the request upload handler.

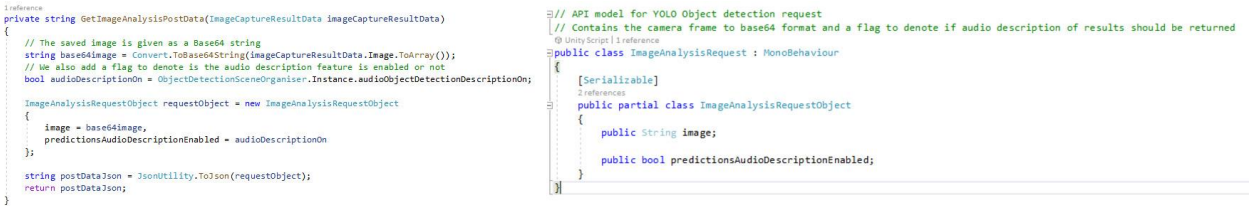


Image 63: Constructing the web request POST data for image analysis

When we receive the response from server, in case of a valid response, as depicted in Image 65, we first have to deserialize the JSON representation into our API response model called "YOLOObjectDetectionAPIResult". This API model contains a list of YOLOAPIPrediction objects, that represent an object detection result. Each of these results, are described using a prediction probability, a tag name for the class that represents the result, and a YOLOAPIBoundingBox object, that contains the result bounding box top-left coordinates in pixel format, along with the width and height of the bounding box.

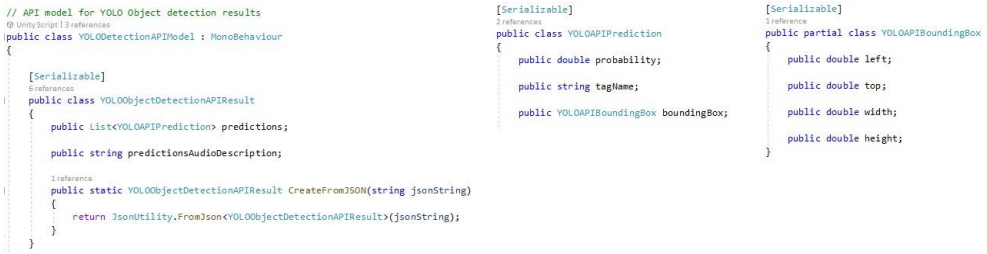


Image 64: The API model for object detection analysis server response

The server response API model has to be converted into a domain model, in order for the object detection result data to be in an appropriate format for placing our object detection results holograms. For this conversion, we use a transformer script, called YoloObjectDetectionApiToDomainTransformer, described in detail in the following section. The domain result model is then passed as an input parameter to the Object Detection Scene Organizer script “LabelDetectedObjects” function, used for visualizing the results with holograms and announcing the audio result descriptive message for Audio scan. In case of a network error during the communication with the server, or in case we receive an invalid response, the object detection process is reset with a sound effect for object detection failure as audio feedback.

```
if (analysisWebRequest.isNetworkError || analysisWebRequest.isHttpError)
{
    Debug.Log("Communication with YOLO server : Network error");
    Debug.Log("Error is " + analysisWebRequest.error);
    ObjectDetectionSceneOrganiser.Instance.ResetDetectionProcessOnError();
}
else
{
    string jsonResponse = analysisWebRequest.downloadHandler.text;

    Debug.Log("response: " + jsonResponse);

    // The response is in JSON format, therefore it needs to be deserialized in our API model
    YOLOObjectDetectionAPIResult analysisResult = new YOLOObjectDetectionAPIResult();
    analysisResult = YOLOObjectDetectionAPIResult.CreateFromJSON(jsonResponse);

    if (analysisResult != null)
    {
        Debug.Log("#Predictions = " + analysisResult.predictions.Count);

        // Convert the result from API to domain model
        YOLOObjectDetectionDomainResult domainResult = YoloObjectDetectionApiToDomainTransformer.TransformYoloObjectDetectionResult(
            analysisResult, imageCaptureResultData.CameraResolution,
            imageCaptureResultData.CameraToWorldMatrix,
            imageCaptureResultData.ProjectionMatrix);

        // Construct the result data object and pass it to scene organiser to label predicted objects
        ObjectDetectionAnalysisResultData analysisResultData = new ObjectDetectionAnalysisResultData(
            domainResult,
            imageCaptureResultData.CameraResolution,
            imageCaptureResultData.CameraToWorldMatrix,
            imageCaptureResultData.ProjectionMatrix
            );

        ObjectDetectionSceneOrganiser.Instance.LabelDetectedObjects(analysisResultData);
    }
    else
    {
        Debug.Log("Predictions is null");
        // Play sound effect for object detection failure
        ObjectDetectionSceneOrganiser.Instance.ResetDetectionProcessOnError();
    }
```

Image 65: Handling the server response for camera frame object detection analysis

6.8.4 Transforming the Object detection server response for visualization

The bounding box coordinates for each object detection result returned from our server belong to the 2D image coordinate system and correspond to a pixel in the camera frame captured and sent for analysis. The holograms we wish to create for each object detection result belong to the 3D world coordinate system. When capturing a camera frame, the photo capture object consists of the native image data and a pair of spatial matrices that indicate where the image was taken. These matrices are the Camera to world matrix and the camera projection matrix. By using these spatial matrices, the 2D image coordinate system and the 3D camera coordinate system can be converted bidirectionally. Since the depth of a point can be calculated by associating it with an environment map provided from HoloLens API, the system can estimate where the point on the image is in the world coordinates.

6.8.4.1 The HoloLens Locatable Camera

HoloLens includes a world-facing camera mounted on the front of the device which enables apps to see what the user sees. Developers have access to and control of the camera just as they would for color cameras on smartphones, portables, or desktops. The same universal windows media capture and windows media foundation APIs that work on mobile and desktop work on HoloLens. Unity has also wrapped these windows APIs to abstract simple usage of the camera on HoloLens for tasks such as taking regular photos and videos (with or without holograms) and locating the camera's position in and perspective on the scene.

When HoloLens takes photos and videos, the captured frames include the location of the camera in the world, as well as the perspective projection of the camera. This allows applications to reason about the position of the camera in the real world for augmented imaging scenarios.

Each image frame (whether photo or video) includes a coordinate system, as well as two important transforms. The "view" transform maps from the provided coordinate system to the camera, and the "projection" maps from the camera to pixels in the image. Together, these transforms define for each pixel a ray in 3D space representing the path taken by the photons that produced the pixel. These rays can be related to other content in the app by obtaining the transform from the frame's coordinate system to some other coordinate system (for example from a stationary frame of reference). To summarize, each image frame provides the following:

- Pixel Data (in RGB/NV12/JPEG/etc. format)
- Three pieces of metadata that make each frame locatable, the Camera Coordinate System, a SpatialCoordinateSystem object that stores the coordinate system of the captured frame the Camera View Transform, a 4x4 matrix that stores the camera's extrinsic transform in the coordinate system, and the Camera Projection Transform, a 4x4 matrix that stores the camera's projection transform.

The projection transform represents the intrinsic properties (focal length, center of projection, skew) of the lens mapped onto an image plane that extends from -1 to +1 in both the X and Y axis [79].

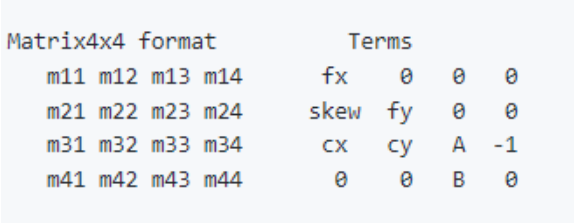


Figure 32: The projection transform and the relation with the camera intrinsic properties [79]

The spatial mapping algorithm in HoloLens estimate the camera extrinsic parameters for all subsequent images and constructs a map of the environment. The detailed process of computing the extrinsic parameters which provides the transformation between the camera coordinate system and the world coordinate system is Microsoft's proprietary software and its algorithms are unpublished [80]. But based on publications by Microsoft researcher [81,82], the spatial mapping technology is based on RGB-D Simultaneous localization and mapping (SLAM) algorithm, where RGB-D data captured by the depth sensor onboard HoloLens are used to construct a map of the environment while simultaneously keeping track of the device's location within it in real time [84].

On HoloLens, the video and still image streams are undistorted in the system's image processing pipeline before the frames are made available to the application (the preview stream contains the original distorted frames). Because only the projection matrix is made available, applications must assume image frames represent a perfect pinhole camera, however the undistortion function in the image processor may still leave an error of up to 10 pixels when using the projection matrix in the frame metadata. In many use cases, this error will not matter, but if we are aligning holograms to real world posters/markers, for example, and we notice an offset at most 10 pixels (roughly 11mm for holograms positioned 2 meters away) this distortion error could be the cause.

The two spatial matrices that work as a bridge between the 2D and 3D coordinate systems are described in detail in the following sections.

6.8.4.2 The Camera to World Matrix

The camera to world matrix mentioned in the previous section is a transformation matrix. Consider a point that is represented as (x, y, z) in the camera space, this is represented as a matrix C = [x, y, z, 1]. The world space coordinates of C are represented as W = [a, b, c, 1] for coordinates (a, b, c). A transformation matrix that works in 3 dimensions is a 4x4 matrix of the form:

$$T_{CamToWorld} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & 0 \\ m_{21} & m_{22} & m_{23} & 0 \\ m_{31} & m_{32} & m_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The relation between the camera space, the world space, and the transformation matrix can be written as:

$$C \times T_{CamToWorld} = W.$$

Given this information we can also write,

$$C \times T_{CamToWorld} \times T^{-1}_{CamToWorld} = W \times T^{-1}_{CamToWorld},$$

which on further simplification becomes

$$C \times I = W \times T^{-1}_{CamToWorld}.$$

Here I is the identity matrix

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Thus, we can write

$$C = W \times T^{-1}_{CamToWorld},$$

which means that the inverse of the transformation matrix can be used to map coordinates from world space to camera space as well. If the inverse of $T_{CamToWorld}$ matrix can be referred to as $T_{WorldToCam}$, we can also write:

$$W \times T_{WorldToCam} = C.$$

6.8.4.3 The Projection Matrix

The projection matrix is a 4x4 matrix that helps in transforming coordinates from the camera coordinate space to image space, that is, the transformation is from a 3D space to a 2D space. The matrix consists of parameters such as the focal length, skew, and center of projection of the camera to assist in its transformational function. Multiplication of a matrix consisting of a camera coordinate (a, b, c) such as C = [a b c 1] yields a result P = [x y 1] such that x and y are the pixel coordinates represented between a range [-1, 1]. If w and h represent the width and height of the image, then we can gain the actual pixel location P_{actual} from the matrix $P_{minimal}$ = [x y] by the following [85]:

$$P_{actual} = \frac{1}{2} P_{minimal} + \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}$$

6.8.4.4 Transformation between the camera and world coordinate system

The Camera to World matrix translates any three-dimensional position in terms of the locatable camera perspective into the three-dimensional coordinate system used by the application in drawing objects into the scene (our mesh is drawn/can be translated, in terms of these coordinates). The projection matrix helps in converting the three-dimensional camera space coordinates into the pixel coordinates of the image.

Given the information above, it is now possible to design a function that would be able to take as input a pixel position and provide a three-dimensional coordinate as output associated with it.

In our case, for each object detection result we receive from our server, we compute the 2D pixel coordinates for the four vertices and the center of the bounding box, based on the top-left coordinate and the width/height information we receive. Firstly, for each 2D

pixel bounding box vertex we have to compute the corresponding 3D world coordinate relative to the camera. Then, we have to transform these to a universal coordinate system defined by the spatial map constructed by HoloLens. An intuitive visualization of this transformation can be seen in Figure 33.

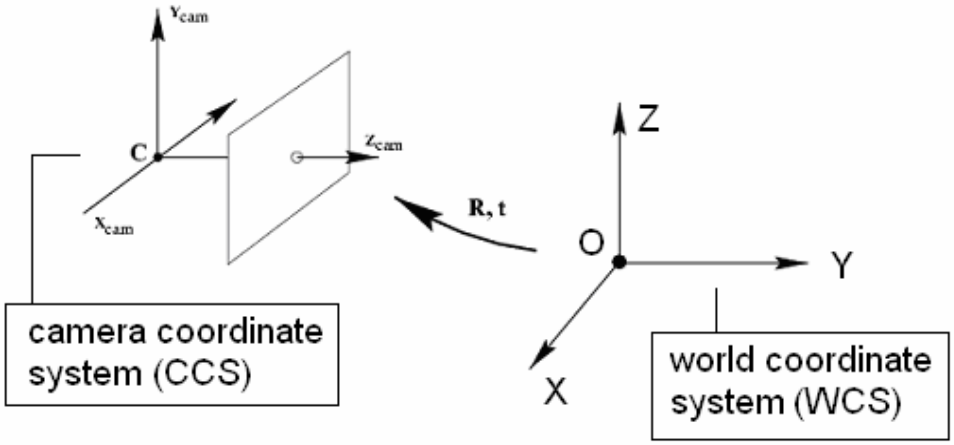


Figure 33: Transformation between world and camera coordinate system [84]

In order to map depth information on RGB frames, we need to find a transformation between the two camera views. This transformation can be expressed through a 4x4 roto-translation matrix, composed by a 3x3 matrix and a 1x3 vector, which hold information about the relative rotation and translation, respectively, between two cameras' frames of reference. In practice, as illustrated in Figure 34, the whole process can be reduced to a series of transformations: from the depth camera 2D projection space to its relative 3D Coordinate System, then to the RGB Coordinate System and finally back to the RGB projection space. One major issue with this approach lies in the temporal misalignment between the recordings of the two sensors. In fact, the HoloLens API does not allow access to the RGB and Depth data streams at the same time [87], leading to a fluctuating mismatch between the frames acquisition time. Because of this, we are forced to consider the sensors as if they were constantly moving with respect to each other. Therefore, the sensors relative position has to be calculated for each pair of frames we want to map between [86].

Initially, we un-project the depth frame pixels to 3D coordinates. The computation is a reverse of camera projection. It is done based on the following equation, where image coordinates are used to calculate the object coordinates in the camera space:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \end{bmatrix},$$

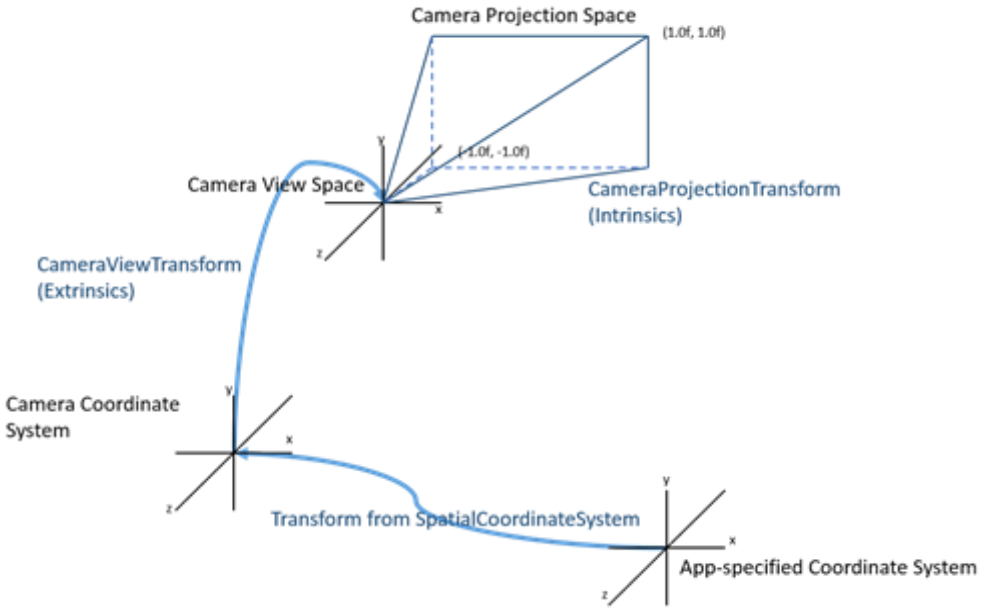


Figure 34: A scheme of the process of locating frames acquired with HoloLens in the real world [79]

where,

- u,v = image coordinates
- X, Y, Z = object coordinates in camera space
- fx, fy, cx, cy = camera intrinsic parameters

For this operation we project a ray from each corner and the center of the bounding box through the perspective center of the camera to compute the location of the bounding box in the camera's coordinate system.

The next step is to calculate the absolute poses of the sensors. We achieve this by combining the Frame to Origin and the Camera View Transformation 4x4 matrices, accessible through the HoloLens API [79].

To summarize, the steps we have to follow for 2D to 3D coordinates mapping operation are presented in the following list:

1. Defined:
   - $P(x_p, y_p)$, a vertex pixel of a detected object bounding box or the center of it, which is a position in the camera frame's space
   - $C(x_c, y_c, z_c)$, the camera position in the world's space, taken at the same time as the frame is acquired

2. P(x_p, y_p) is transformed from the frame space, to the world space, obtaining P'(x_p, y_p, z_p)
3. A ray, as seen in Figure 35, starting from C(x_c, y_c, z_c) and passing through P'(x_p, y_p, z_p) is traced. The intersection between the ray and the environmental mesh identifies a point I(x_i, y_i, z_i), that is an approximation of the object's location in the world space. This step is easily performed by using Unity 3D Ray Tracing and Collider functionalities.
4. Points from 1 to 3 are repeated for each object detection result.

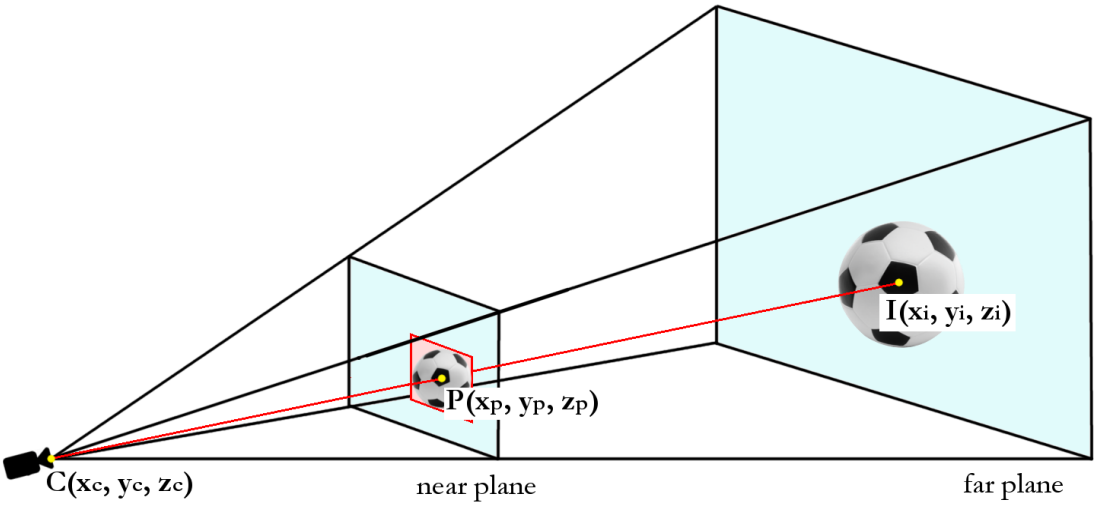


Figure 35: 2D to 3D Spatial Mapping [89]

6.8.4.5 The Yolo Object Detection API to Domain Transformer

Let us now describe the way the above transformation operations are applied to our transformer script for transforming the API server response to a domain response that is suitable for our object detection results visualization by holograms placing.

The transformer function takes as input parameter the API server response model, along with the camera resolution object and the spatial matrices cameraToWorldMatrix and projectionMatrix as retrieved from our camera frame capture step. For our domain model a new list of "YOLODomainPrediction" objects is initialized, related to the domain model for each object detection result retrieved. Looping through the API predictions list, we first have to set the 2D pixel coordinates for the center of the result bounding box and for each of its four vertices, given the top-left pixel coordinate and the width/height that are returned from the YOLO algorith applied server side. This

operation is executed in "SetCoordinatesOfPrediction" method, defined in YOLOAPIPrediction class, as seen in Image 66.

```
1 reference
public void SetCoordinatesOfPrediction()
{
    float centerXCoordinate = (float)(boundingBox.left + (0.5 * boundingBox.width));
    float centerYCoordinate = (float)(boundingBox.top + (0.5 * boundingBox.height));
    centerCoordinates = new Vector2(centerXCoordinate, centerYCoordinate);

    double boundingBoxRight = boundingBox.left + boundingBox.width;
    double boundingBoxBottom = boundingBox.top + boundingBox.height;

    topLeftCoordinates = new Vector2((float)boundingBox.left, (float)boundingBox.top);
    topRightCoordinates = new Vector2((float)boundingBoxRight, (float)boundingBox.top);
    bottomLeftCoordinates = new Vector2((float)boundingBox.left, (float)boundingBoxBottom);
    bottomRightCoordinates = new Vector2((float)boundingBoxRight, (float)boundingBoxBottom);
}
```

Image 66: Setting the 2D coordinates for the center and four vertices of the result bounding box

Since the 2D pixel coordinates for an object detection result are computed, we now have to convert those coordinates from pixel space to 3D world space as described in the previous sections. For this conversion task we have introduced a separate utility transformer class, defined as "CoordinateTransfer". The "Image Position to World Position" function takes a 2D vector pixel position, the camera resolution object, and the necessary spatial maps (camera to world and projection matrices) and in case the conversion is successful, the 3D world position result is extracted to a 3D vector passed as the last input parameter. This function returns a Boolean value that indicates if the 3D position extracted is valid and the conversion task is completed successfully or not. The 2D to 3D coordinates conversion is applied for the center and the four vertices of the bounding box of an object detection result. If all 3D word positions are valid, a new "YOLODomainPrediction" object is constructed with these 3D coordinates needed for placing the label and bounding box holograms, along with the prediction probability and class name of the result.

Once the domain list of the predictions is filled in case an audio description result descriptive message is available from our server, which applied for Audio scan, we want to enhance this message with information regarding the distance of each detected object from the user. For this reason, we call the "Get Object Detection Results Distances Description" function, which computes the distance in meters of each result from the camera/user position, as depicted in Image 68.

```
1 reference
public static YOLOObjectDetectionDomainResult TransformYoloObjectDetectionResult(YOLOObjectDetectionAPIResult apiResult,
    Resolution cameraResolution, Matrix4x4 cameraToWorldMatrix, Matrix4x4 projectionMatrix)
{

    IList<YOLODomainPrediction> domainPredictionsList = new List<YOLODomainPrediction>();

    IList<YOLOAPIPrediction> apiPredictionsList = apiResult.predictions;

    foreach (var apiPrediction in apiPredictionsList)
    {
        // Set the center of prediction coordinates in 2D space
        apiPrediction.SetCoordinatesOfPrediction();

        // Get the center of prediction in 3D space
        bool isCenterPositionValid = CoordinateTransfer.ImagePositionToWorldPosition(apiPrediction.centerCoordinates,
            cameraResolution, projectionMatrix, cameraToWorldMatrix, out Vector3 centerWorldPosition);

        // Get the four corners of the prediction bounding box in 3D space
        bool isTopLeftPositionValid = CoordinateTransfer.ImagePositionToWorldPosition(apiPrediction.topLeftCoordinates,
            cameraResolution, projectionMatrix, cameraToWorldMatrix, out Vector3 topLeftWorldPosition);
        bool isTopRightPositionValid = CoordinateTransfer.ImagePositionToWorldPosition(apiPrediction.topRightCoordinates,
            cameraResolution, projectionMatrix, cameraToWorldMatrix, out Vector3 topRightWorldPosition);
        bool isBottomLeftPositionValid = CoordinateTransfer.ImagePositionToWorldPosition(apiPrediction.bottomLeftCoordinates,
            cameraResolution, projectionMatrix, cameraToWorldMatrix, out Vector3 bottomLeftWorldPosition);
        bool isBottomRightPositionValid = CoordinateTransfer.ImagePositionToWorldPosition(apiPrediction.bottomRightCoordinates,
            cameraResolution, projectionMatrix, cameraToWorldMatrix, out Vector3 bottomRightWorldPosition);

        // Check if all 3D positions are valid, then add a domain result to the list
        if (isCenterPositionValid && isTopLeftPositionValid && isTopRightPositionValid && isBottomLeftPositionValid && isBottomRightPositionValid)
        {
            YOLODomainPrediction domainPrediction = new YOLODomainPrediction(apiPrediction.probability, apiPrediction.tagName, centerWorldPosition,
                topLeftWorldPosition, topRightWorldPosition, bottomLeftWorldPosition, bottomRightWorldPosition);
            domainPredictionsList.Add(domainPrediction);
        }

    }

    YOLOObjectDetectionDomainResult result = new YOLOObjectDetectionDomainResult(domainPredictionsList)
    {
        // Set the audio description result from server response
        AudioPredictionsDescription = apiResult.predictionsAudioDescription + GetObjectDetectionResultsDistancesDescription(domainPredictionsList)
    };

    return result;
}
```

Image 67: The Yolo Object Detection API to Domain Transformer

```
1 reference
private static string GetObjectDetectionResultsDistancesDescription(IList<YOLODomainPrediction> domainPredictionsList)
{
    string distancesText = "";

    Vector3 headPosition = Camera.main.transform.position;

    foreach (var domainPrediction in domainPredictionsList)
    {
        float distance = Vector3.Distance(domainPrediction.DetectionCenterWorldPosition, headPosition);
        float distanceWithTwoDecimals = (float)Math.Round(distance * 100f) / 100f;

        distancesText += " The " + domainPrediction.TagName + " is " + distanceWithTwoDecimals.ToString() + " meters far away.";
    }

    return distancesText;
}
```

Image 68: Enhancing the audio descriptive message of object detection results with distances from the user

Finally, let us describe in detail the steps for transforming a 2D pixel coordinate to a 3D world coordinate, based on the Microsoft official documentation for HoloLens Locatable Camera [79]. Initially, we have to divide the pixel's x and y components by the camera frame width and height respectively. The width and height information can be derived from the camera resolution object we have selected for our capture process. We then

convert the point to normalized device coordinates, where the values in range [-1,1]. Based on these normalized device coordinates, we construct a 3D vector "imagePosProjected" that represent the point in Camera projection space, as seen in Figure 34. We set the point's z value from the depth data, converted to the range [0,1].

```
5 references
public static class CoordinateTransfer
{

    // Get an image 2D (x,y) position and return the position to world coordinates on positionToWorldCoordinates vector
    // Returns true if a valid position is returned to the result vector, false otherwise
    5 references
    public static bool ImagePositionToWorldPosition(Vector2 pixelCoordinates, Resolution cameraResolution,
        Matrix4x4 projectionMatrix, Matrix4x4 cameraToWorldMatrix, out Vector3 positionToWorldCoordinates)
    {
        // The projection transform represents the intrinsic properties of the lens mapped onto an image plane
        // that extends from -1 to +1 in both the X and Y axis.
        Vector2 imagePosZeroToOne = new Vector2(pixelCoordinates.x / cameraResolution.width, 1 - pixelCoordinates.y / cameraResolution.height);
        Vector2 imagePosProjected2D = imagePosZeroToOne * 2.0f - new Vector2(1.0f, 1.0f);  // -1 to 1 space

        Vector3 imagePosProjected = new Vector3(imagePosProjected2D.x, imagePosProjected2D.y, 1);
        Vector3 cameraSpacePos = UnProjectVector(projectionMatrix, imagePosProjected);

        //worldSpaceRayPoint1 is cameraPosition, which is equivalent to the starting point of Ray
        //worldSpaceRayPoint2 is equivalent to the end point of Ray.
        Vector3 cameraLocationWorldSpace = cameraToWorldMatrix.MultiplyPoint(Vector3.zero); // camera location in world space
        Vector3 rayPointWorldSpace = cameraToWorldMatrix.MultiplyPoint(cameraSpacePos); // ray point in world space

        // We call Physics.Raycast to find the collision point between Ray and the real scene, that is, the location of the target object
        // Do a ray cast to the spatial map from the camera position through the location we calculated,
        // basically shooting 'through' the picture for the real object.

        if (Physics.Raycast(cameraLocationWorldSpace, rayPointWorldSpace - cameraLocationWorldSpace, out RaycastHit hit, 20f, SpatialMapping.PhysicsRaycastMask))
        {
            positionToWorldCoordinates = hit.point;
            return true;
        }
        else
        {
            Debug.Log("Collision with the real scene failed!");
        }

        positionToWorldCoordinates = Vector3.zero;
        return false;
    }
```

Image 69: Coordinate transfer from 2D pixel space to 3D world space for a point, based on the Microsoft documentation in [79]

The unprojection operation for getting the point in the camera 3D space takes place by calling the "UnProjectVector" function based on the documentation in [79], as seen in Image 79, given the projection matrix and the point in camera projection space as input parameters. The next step is to get the camera location (Vector3.zero) and the point position from the 3D camera space to the 3D world space. This is achieved by multiplying the 3D vectors with the Camera to World spatial map.

```
1 reference
private static Vector3 UnProjectVector(Matrix4x4 proj, Vector3 to)
{
    Vector3 from = new Vector3(0, 0, 0);
    var axsX = proj.GetRow(0);
    var axsY = proj.GetRow(1);
    var axsZ = proj.GetRow(2);
    from.z = to.z / axsZ.z;
    from.y = (to.y - (from.z * axsY.z)) / axsY.y;
    from.x = (to.x - (from.z * axsX.z)) / axsX.x;
    return from;
}
```

Image 70: Vector unprojection function based on the Microsoft documentation in [79]

The final step we need to take in order to return our final point world position 3D vector is to apply Z- Filtering. This procedure takes as input the camera position and the point we are interested in to be judged as occluded in view or not. The layer mask is set such that only Unity layer 31 is considered for collision with the ray. Layer 31 is the layer the spatial mapping mesh is assigned to by default. If a hit happens and the collision occurred in close proximity to the intended point, the method returns a true value. Else, the vertex is flagged as occluded by returning false. For this purpose, we apply a raycast starting from the camera with a direction pointing to the point of interest, in a maximum distance of 20 meters, considering the guidelines for the user's field of view wearing a HoloLens device. If the position of collision on the mesh in view, matches with the vertex to which the ray cast was intended, the vertex is safe to record. The hit point of the raycast is returned as our final 3D output vertex of the pixel position in 3D world space.

6.8.5 Object detection results visualization

After the successful transformation of the server response as described in the previous section, the domain detection result data are sent as an input parameter of “LabelDetectedObjects” function. This member function belongs to our core Object Detection Scene Organizer script and is responsible for placing the holograms for each detection result, as well as for triggering the audio description message for the detected results in case the user has selected the Audio scan feature of the application.

```
// Used to place the detected objects labels
// At then end, reset the capture process to allow the user to capture another image.
1 reference
public virtual void LabelDetectedObjects(ObjectDetectionAnalysisResultData detectionResultData)
{
    // First, clear all the prediction labels and bounding boxes placed from previous analysis result
    ClearPreviousPredictionLabels();
    ClearPreviousPredictionBoundingBoxes();

    IList<YOLODomainPrediction> predictions = detectionResultData.DetectionDomainResult.Predictions;

    if (predictions.Count > 0)
    {
        PlaySoundForObjectDetectionSuccess();
    }
    else
    {
        PlaySoundForObjectDetectionFailure();
    }

    foreach (var prediction in predictions)
    {
        predictionLabelObjectsList.Add(CreatePredictionResultLabel(prediction.TagName, prediction.DetectionCenterWorldPosition));
        predictedBoundingBoxObjectsList.Add(CreatePredictionResultBoundingBox(prediction));
    }

    //If audio description feature is enabled, speak the detection result prediction message
    SpeakObjectDetectionPredictionResultMessage(detectionResultData.DetectionDomainResult);

    ResetDetectionProcess();
}
```

Image 71: Labeling the object detection results

Before instantiating and placing a new set of holograms for the current object detection results, we first have to clear any holograms placed from a previous analysis result. For each object detection result, we place a label tooltip hologram in the center of the bounding box, along with the bounding box hologram that encloses the detected object. The label and bounding box hologram game objects from a single scan result, are kept in a list as properties of our scene organizer script. As depicted in Image 72, during the holograms clearing process, each label and bounding box hologram game object are destroyed and freed up from memory, before clearing the game objects property lists.

```
// For all previously created labels, destroy the game object and clear the list
1 reference
private void ClearPreviousPredictionLabels()
{
    foreach (var label in predictionLabelObjectsList)
    {
        Destroy(label);
    }
    predictionLabelObjectsList.Clear();
}

// For all previously created bounding boxes, destroy the game object and clear the list
1 reference
private void ClearPreviousPredictionBoundingBoxes()
{
    foreach (var boundingBox in predictedBoundingBoxObjectsList)
    {
        Destroy(boundingBox);
    }
    predictedBoundingBoxObjectsList.Clear();
}
```

Image 72: Clearing of any holograms placed for a previous object detection analysis

Subsequently, we play a sound effect related to the object detection scan status as user audio feedback. In case we have at least one object detected, we play a success sound effect, otherwise we play a failure sound effect with the help of our Audio effects player.

The next step is to place a tooltip label and bounding box hologram for each object detection result of the latest analysis performed, as depicted in detail in Image 73.

The prediction tooltip label hologram placing is handled by "Create Prediction Result Label" function, which takes the prediction result class name as a parameter, along with a 3D world coordinate vector of the bounding box center to place the hologram. The label game object is instantiated from the “predictionLabelObject” configuration parameter of the script, as seen in Image 41, which is a Simple Line Tooltip prefab asset provided by the MRTK library. From this game object we get access to the Tooltip component in order to perform the necessary configurations. We set the tooltip's text to be the prediction class name. Furthermore, we set the tooltip’s position in world space to be equal to the center of the bounding box parameter. Finally, we set the parent game object of the tooltip to be the “predictionObjectsContainer” configuration parameter of the script, which is set to the “PredictionObjectsContainer” game object for

Visual scan and “AudioPredictionObjectsContainer” game object for Audio scan respectively.

The bounding box hologram is handled by "Create Prediction Result Bounding Box" function, that takes a domain YOLODomainPrediction object as a parameter. The bounding box game object is instantiated from the “predictionBoundingBoxObject“ configuration parameter of the script, which is our custom BoundingBox prefab asset. From this game object we get access to the BoundingBox component to call our custom initialize function of the asset to draw the bounding box line renderer given an array of the four bounding box vertices in 3D world coordinates. Finally, we set the parent game object of the bounding box to be the “predictionObjectsContainer” configuration parameter of the script, as described for the label holograms above.

```
// Create the result label tooltip gameobject
// Based on https://github.com/microsoft/MixedRealityToolkit-Unity/blob/265a0ea3621c84b88e4b02e7fcd1b5682afef56a/Assets/MRTK/SDK/
1 reference
private GameObject CreatePredictionResultLabel(string predictionResultClassName, Vector3 predictionResultLocation)
{
    var labelObject = Instantiate(predictionLabelObject);
    var toolTip = labelObject.GetComponent<ToolTip>();

    toolTip.ToolTipText = predictionResultClassName;
    // point of tooltip rectangle , slightly up from the prediction location
    toolTip.transform.position = predictionResultLocation;
    toolTip.transform.parent = predictionObjectsContainer.transform;

    return labelObject;
}

1 reference
private GameObject CreatePredictionResultBoundingBox(YOLODomainPrediction prediction)
{
    var boundingBoxObject = Instantiate(predictionBoundingBoxObject);
    var boundingBox = boundingBoxObject.GetComponent<BoundingBox>();

    // Set the bounding box world coordinates
    boundingBox.Initialize(prediction.BoundingBoxCornersNumber, prediction.DetectionBoundingBoxCornerWorldCoordinates);

    boundingBox.transform.parent = predictionObjectsContainer.transform;

    return boundingBoxObject;
}

1 reference
private void SpeakObjectDetectionPredictionResultMessage(YOLOObjectDetectionDomainResult detectionResultModel)
{
    if (audioObjectDetectionDescriptionOn && !string.IsNullOrEmpty(detectionResultModel.AudioPredictionsDescription))
    {
        TextToSpeech textToSpeech = predictionObjectsContainer.GetComponent<TextToSpeech>();
        var detectionResultMsg = string.Format(detectionResultModel.AudioPredictionsDescription, textToSpeech.Voice.ToString());
        textToSpeech.StartSpeaking(detectionResultMsg);
    }
}
```

Image 73: Placing of tooltip and bounding box holograms for each object detection result

The last action before resetting the object detection process, is to trigger the object detection descriptive audio message that announces information related to each detection result and its relative position to the user's view, along with a distance from the user. Using the TextToSpeech component added to the prediction objects container, we call the "StartSpeaking" method to provide the audio feedback for Audio scan.

In the following images we can see our object detection mixed reality application visualization results in practice:

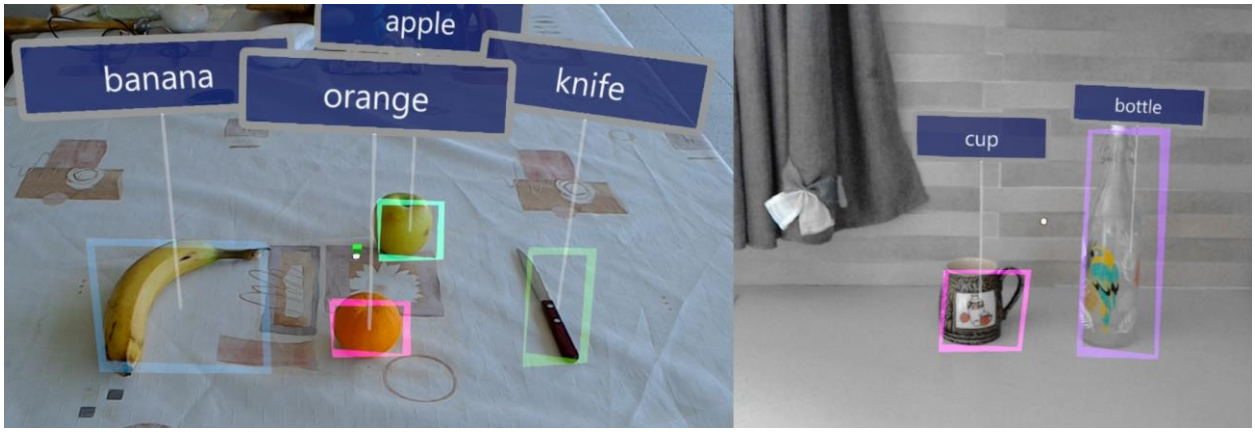


Image 74: Our Mixed Reality application in practice, detecting a banana, orange, apple, knife, cup, bottle

Image 75: Our Mixed Reality application in practice, detecting a sofa, remote, cell phone

Image 76: Our Mixed Reality application in practice, detecting a bed and dog

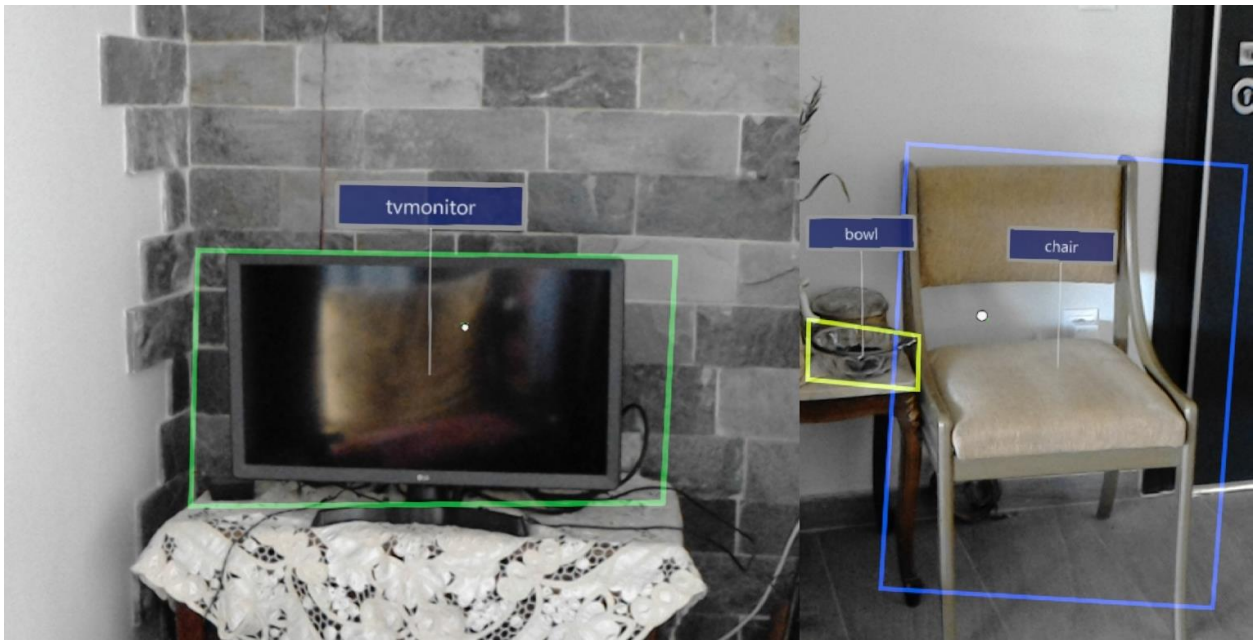


Image 77: Our Mixed Reality application in practice, detecting a tv monitor, chair, bowl

Image 78: Our Mixed Reality application in practice, detecting a chair, handbag, backpack

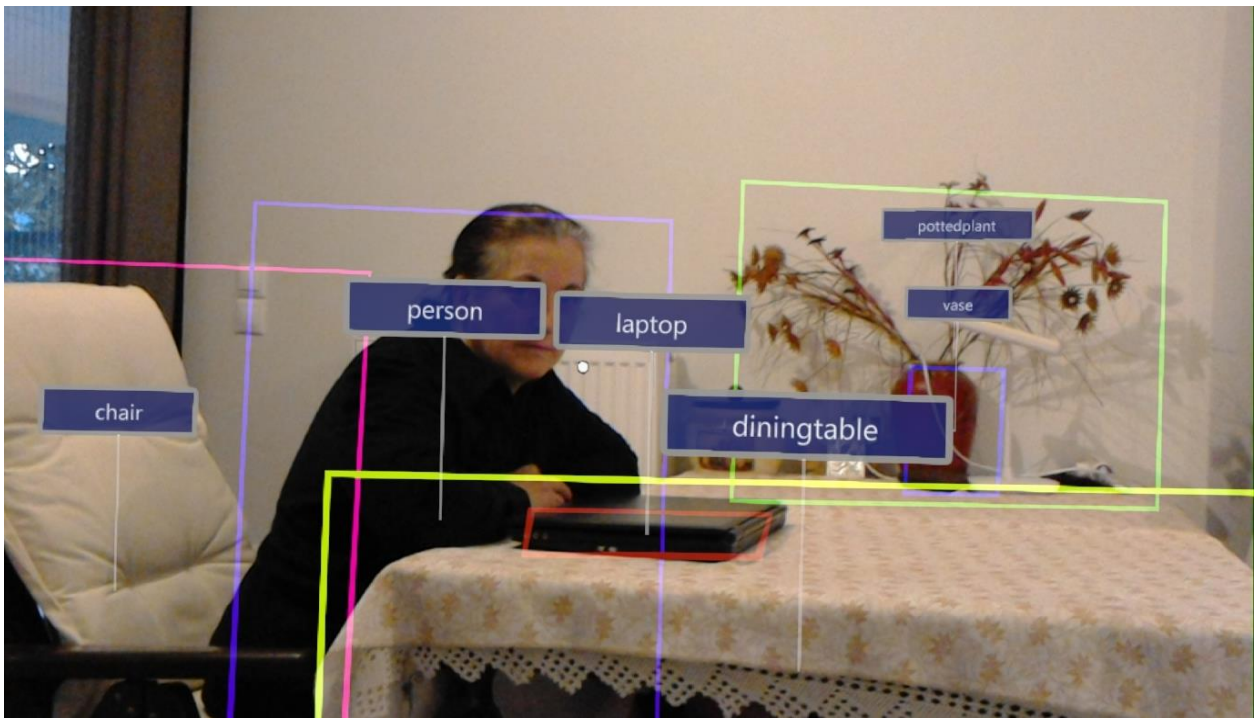


Image 79: Our Mixed Reality application in practice, detecting a chair, person, laptop, dining table, vase, and potted plant

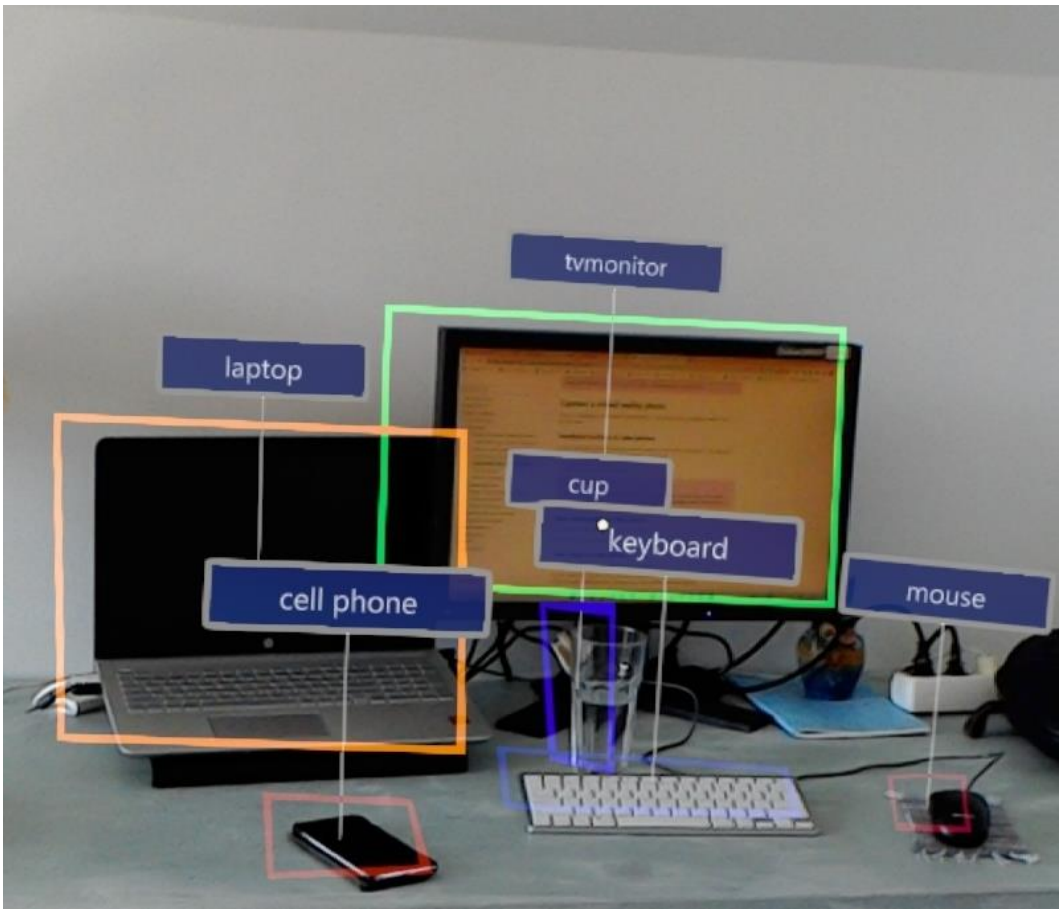


Image 80: Our Mixed Reality application in practice, detecting a laptop, cell phone, tv monitor, cup, keyboard, and mouse

7. OUR NVIDIA JETSON TX2 SERVER APPLICATION

Our development server application for object detection and camera frames similarity is configured to run inside a docker container on NVIDIA Jetson TX2 module using Flask framework. We have downloaded and installed NVIDIA L4T ML docker container [32], that includes TensorFlow, PyTorch, JupyterLab, and other popular ML and data science frameworks such as numpy and OpenCV pre-installed in a Python 3.6 environment. We use a pretrained YOLOv3 model trained on MS COCO dataset. Object detection and frame similarity functions were developed with OpenCV library. In this chapter we describe in detail the configuration steps for running our NVIDIA docker container and for starting our server application from it. Detailed information regarding our Python scripts for object detection and camera frames similarity used by our RESTFul web service are also given.

7.1 NVIDIA L4TML Docker container configuration

The l4t-ml docker image is a Machine Learning container offered by NVIDIA for Jetson and JetPack that contains TensorFlow, PyTorch, JupyterLab, and other popular ML and data science frameworks such as scikit-learn, scipy, and Pandas pre-installed in a Python 3.6 environment. It also includes numpy and OpenCV needed for our object detection and frame similarity scripts pre-installed. These containers support various releases of JetPack for Jetson TX2, including Jetpack version 4.5 which is used for our project. The official page of l4t-ml container [32] provides guidelines for running the container, connecting to JupyterLab Server and mounting directories from the host Jetson device. Based on these guidelines we first pull the correct container version based on the JetPack version we have installed in our Jetson device.

For starting an interactive session in the container, we have created a script as seen in Image 81, with certain configuration steps.

To mount scripts and data from our Jetson's filesystem to run inside the container, we use Docker's -volume flag when starting our Docker instance.

Additionally, a JupyterLab server instance is automatically started along with the container. We can connect to it by navigating our browser to http://localhost:8888 or substitute the IP address of the Jetson device if we wish to connect from a remote host, with the Jetson in headless mode. For this reason, we use -publish flag when starting our Docker instance.

```
if [ -z "${L4T_ML_ID}" ]; then
    echo "Creating new l4t-ml container."
    xhost +
    docker run --runtime nvidia -it --rm \
        --network bridge \
        --name ${L4T_ML_NAME} \
        --volume ${HOST_DATA_DIR}:${CONTAINER_DATA_DIR}:rw \
        --volume /tmp/.X11-unix:/tmp/.X11-unix \
        --volume /tmp/argus_socket:/tmp/argus_socket \
        --device /dev/video0 \
        --env DISPLAY=unix${DISPLAY} \
        --publish 8080:8888/tcp \
        --publish 8081:8081/tcp \
        nvcr.io/nvidia/l4t-ml:${CONTAINER_TAG}
else
    echo "Found l4t-ml container: ${L4T_ML_ID}."
    # Check if the container is already running and start if necessary.
    if [ -z `docker ps -qf "name=^/${L4T_ML_NAME}$"` ]; then
        xhost +local:${L4T_ML_ID}
        echo "Starting and attaching to ${L4T_ML_NAME} container..."
        docker start ${L4T_ML_ID}
        docker attach ${L4T_ML_ID}
    else
        echo "Found running ${L4T_ML_NAME} container, attaching bash..."
        docker exec -it ${L4T_ML_ID} bash
    fi
fi
```

Image 81: Our L4TML Docker container configuration script

By default, when we create or run a container using docker create or docker run, it does not publish any of its ports to the outside world. To make a port available to services outside of Docker, or to Docker containers which are not connected to the container's network, we can use the --publish or -p flag. This creates a firewall rule which maps a container port to a port on the Docker host to the outside world.

For the JupyterLab server used for development purposes, we have mapped TCP port 8888 in the container to port 8080 in the Docker host, with --publish 8080:8888/tcp.

The same publish configuration is used for being able to start our server application for our project, in order to be able to access our RESTFul web services outside the container, using the Jetson device IP address. For our Flask web application, we have mapped TCP port 8081 in the container, to port 8081 in the in the Docker host, with --publish 8081:8081/tcp.

Finally, in case the container is found already running, our script only attaches a bash to it, so we can enter the container and start our development server application.

7.2 Our Flask server application

To run our server application, we have created a Python app.py script.

First, we import the necessary packages, including our methods for object detection and camera frames similarity from separate scripts, that will be described in detail in the following sections. From flask packages we import the Flask class. An instance of this class will be our Web Server Gateway Interface (WSGI) application. We also import Flask jsonify, a functionality within Python's capability to convert a JSON output into a response object with application/json mimetype by wrapping up a json.dumps( ) function for adding the enhancements. Along with the conversion of JSON to an output response, this function helps in conversion of multiple arguments to array or multiple arguments into dictionary.

```
from flask import Flask,jsonify,request
from yolo_detection_images import detectObjects
from frames_comparison import framesComparison
import os

app = Flask(__name__)

@app.route('/HoloLensObjectDetection/detectObjects', methods=['POST'])
def detectObjectsFromImage():
    post_params = request.get_json()
    image_base64 = post_params['image']
    shouldReturnAudioDescr = post_params['predictionsAudioDescriptionEnabled']
    results = detectObjects(image_base64, shouldReturnAudioDescr)
    return jsonify(results)

@app.route('/HoloLensObjectDetection/checkFramesSimilarity', methods=['POST'])
def checkForFramesSimilarity():
    post_params = request.get_json()
    image_base64_previous_frame = post_params['previous_frame_image']
    image_base64_current_frame = post_params['current_frame_image']
    result = framesComparison(image_base64_previous_frame, image_base64_current_frame)
    return jsonify(result)


if __name__ == "__main__":
    port = int(os.environ.get("PORT", 8081))
    app.run(debug=True,host='0.0.0.0',port=port)
```

Image 82: Our server application script

Next, we create an instance of Flask class with app = Flask(__name__). The first argument is the name of the application's module or package. __name__ is a convenient shortcut for this that is appropriate for most cases. This is needed so that Flask knows where to look for resources such as templates and static files.

We then use the route decorator to tell Flask what URLs should trigger our functions.

When we receive a POST request under the “/HoloLensObjectDetection/detectObjects” path, we call the “detectObjectsFromImage” script method, for applying object detection on a camera frame sent from HoloLens. We extract the “image” POST request parameter, which contains the HoloLens camera frame native data in Base64 format. We also extract the “predictionsAudioDescriptionEnabled” POST request parameter, which denotes if our script function should return the object detection results in an audio descriptive message. These parameters are passed as input to the “detectObjects” function defined in a separate script for object detection. The result is then transformed to a JSON response to be sent back to our HoloLens device for handling.

Similarly, when we receive a POST request from HoloLens that targets the path “/HoloLensObjectDetection/checkFramesSimilarity”, we call our script method for camera frames comparison, called “checkForFramesSimilarity”. We extract two POST request parameters, named “previous_frame_image” and “current_frame_image”, that correspond to the native image data in Base64 format for the preceding and current camera frame respectively. These request parameters are sent as an input to our dedicated script for frames comparison, which returns a result that is transformed to JSON response for our HoloLens device.

In the main function of our server application script, we set that our web application will be publicly available on port 8081 of the container, which is mapped to the Jetson’s host device 8081 port and start our server.

7.2.1 Our server script for applying YOLO Object detection

For our dedicated script to apply the YOLOv3 object detection algorithm on an image sent from our HoloLens, we first need to import our required packages which are numpy, cv2 for OpenCV and base64 for converting out base64 image input data.

We first define some configuration thresholds for our detection process, we set a threshold to accept detected results with confidence greater than 50% and a threshold for YOLO algorithm non-max suppression process to be 0.3.

Since we use a pretrained YOLOv3 model for our object detection, we have downloaded and saved the configuration parameters and pretrained weights files from the official YOLO algorithm page [53]. We load the parameter and weights in “modelConfiguration” and “modelWeights” variables respectively. We also load the COCO dataset labels, on which our model was trained on, and construct a “labels” array

with the detectable classes. The next step is to load our YOLO object detector using OpenCV “readNetFromDarknet” function, passing the model configuration and weights. Since our DNN is now set up, we convert our input image from a Base64 image to an OpenCV image and extract its dimensions in a vector. From this OpenCV image, we construct a blob with a 1/255 scale factor and perform a forward pass of the YOLO object detector, which gives us the bounding boxes and the associated probabilities in the “layerOutputs” variable. We then initialize a set of lists for the bounding boxes, confidence scores, class IDs and detection result centers.

```
import numpy as np
import base64
import cv2

def detectObjects(base64_img_data, shouldReturnAudioDescription):
    confidenceThreshold = 0.5
    NMSThreshold = 0.3

    modelConfiguration = 'cfg/yolov3.cfg'
    modelWeights = 'yolov3.weights'

    # load the COCO class labels our YOLO model was trained on
    labelsPath = 'coco.names'
    labels = open(labelsPath).read().strip().split('\n')

    # load our YOLO object detector trained on COCO dataset (80 classes)
    net = cv2.dnn.readNetFromDarknet(modelConfiguration, modelWeights)

    image = dataBase64ToCVImage(base64_img_data)
    (H, W) = image.shape[:2]

    #Determine only the *output* layer names that we need from YOLO
    layerName = net.getLayerNames()
    layerName = [layerName[i[0] - 1] for i in net.getUnconnectedOutLayers()]

    # construct a blob from the input frame and then perform a forward
    # pass of the YOLO object detector, giving us our bounding boxes
    # and associated probabilities
    blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416), swapRB = True, crop = False)
    net.setInput(blob)
    layersOutputs = net.forward(layerName)

    # initialize our lists of detected bounding boxes, confidences and class IDs, respectively
    boxes = []
    confidences = []
    classIDs = []
    centers = []
```

Image 83: Our script for YOLO Object detection - Initialization steps

Looping through each of the output layers of our YOLO object detection network, for each of the detection results, we extract the class ID and probability score. We filter out all weak predictions by ensuring that the detected probability is greater than the minimum probability threshold we have set. In case the detection result probability is

accepted, we scale the bounding box coordinates back relative to the actual size of the input image. Keeping in mind that YOLO returns the center coordinates of the bounding box followed by the boxes' width and height, we use these coordinates to derive the top and left corner of the bounding box. We update our lists with the bounding box coordinates, confidences, and class IDs for each object detection result. After looping through all the detected objects returned, we apply Non-Maxima Suppression to suppress the weak overlapping bounding boxes. The "detectionNMS" variable holds the final output of our model.

```
# loop over each of the layer outputs
for output in layersOutputs:
    # loop over each of the detections
    for detection in output:
        # extract the class ID and confidence (i.e., probability) of the current object detection
        scores = detection[5:]
        classID = np.argmax(scores)
        confidence = scores[classID]
        # filter out weak predictions by ensuring the detected probability
        # is greater than the minimum probability
        if confidence > confidenceThreshold:
            # scale the bounding box coordinates back relative to the
            # size of the image, keeping in mind that YOLO actually
            # returns the center (x, y)-coordinates of the bounding
            # box followed by the boxes' width and height
            box = detection[0:4] * np.array([W, H, W, H])
            (centerX, centerY,  width, height) = box.astype('int')

            # use the center (x, y)-coordinates to derive the top and
            # and left corner of the bounding box
            x = int(centerX - (width/2))
            y = int(centerY - (height/2))

            # update our list of bounding box coordinates, confidences, and class IDs
            boxes.append([x, y, int(width), int(height)])
            confidences.append(float(confidence))
            classIDs.append(classID)
            centers.append((centerX, centerY))

#Apply Non Maxima Suppression to suppress weak, overlapping bounding boxes
detectionNMS = cv2.dnn.NMSBoxes(boxes, confidences, confidenceThreshold, NMSThreshold)
```

Image 84: Looping over the detection results of our YOLOv3 model

The final output from "detectionNMS" variable has to be converted to a JSON object for our REST API call response. In case the variable size is zero, no objects are detected. Otherwise, since the output of the model is in a form of a nested list, we use the "flatten" function to convert it into a single list. For every detection result, we construct a detection JSON object that contains the detection probability, the class name using the labels array from our dataset file, and a bounding box JSON object, that contains the

left-top coordinates in pixel form, along with the width and height of the bounding box that encloses the detection result.

```
# Output json object with predictions array
outputs = {}
outputs['predictions'] = []

# Used for audio predictions result
texts = []

if len(detectionNMS)>0:
    for i in detectionNMS.flatten():

        # Detection json object for each result
        detection = {}

        detection['tagName'] = labels[classIDs[i]]
        detection['probability'] = confidences[i]

        boundingbox = {}
        boundingbox['left'] = boxes[i][0]
        boundingbox['top'] = boxes[i][1]
        boundingbox['width'] = boxes[i][2]
        boundingbox['height'] = boxes[i][3]
        detection['boundingBox'] = boundingbox

        outputs['predictions'].append(detection)

        # If the audio description is enabled, get the prediction in audio output
        if shouldReturnAudioDescription:
            centerX, centerY = centers[i][0], centers[i][1]
            texts.append(getPredictionAudioOutput(W, H, centerX, centerY, labels[classIDs[i]]))

    if shouldReturnAudioDescription:
        outputs['predictionsAudioDescription'] = ' '.join(texts)

else:
    if shouldReturnAudioDescription:
        outputs['predictionsAudioDescription'] = 'No objects detected, please try a different view.'

return outputs
```

Image 85: Constructing the JSON response with our YOLO object detection results

If the audio description feature is enabled, we also have to construct a descriptive text with the object's class name and position within the image. For this purpose, we use "getPredictionAudioOutput" function. Based on the portion of the image width and height that the object detection center lies into, we extract the relative position text. An example result would be “There is a book at the top right of the view”. The object detection texts for all detected objects are joined together to form the audio message of the detection that we return to the HoloLens device.

```
def getPredictionAudioOutput(imageWidth, imageHeight, centerX, centerY, label):
    # find positions
    if centerX <= imageWidth/3:
        W_pos = "left "
    elif centerX <= (imageWidth/3 * 2):
        W_pos = "center "
    else:
        W_pos = "right "

    if centerY <= imageHeight/3:
        H_pos = "top "
    elif centerY <= (imageHeight/3 * 2):
        H_pos = "middle "
    else:
        H_pos = "bottom "

    text = 'There is a '+ label+ ' at the '+ H_pos + W_pos + ' of the view.'
    return text
```

Image 86: The function for constructing the audio description message for the object detection

7.2.2 Our server script for camera frames comparison

In order to determine if two camera frames are similar or not, we compute the correlation coefficient of the grayscale histograms for the two camera frame images. If the value of the correlation is greater than a threshold we have defined, the frames are considered as similar. We have set the frames correlation threshold to be equal to 0.80. We chose this value, as we do not want our filter for considering two frames unsimilar to be very sensitive. If two camera frames have significant differences that indicate that the user has been moved or has stayed still but the user's scene has changed, we consider the two frames different to fire the object detection process for the current frame.

The first step of our computations is to convert the camera frames images from Base64 to OpenCV images and then convert them to grayscale using the OpenCV "cvtColor" function passing COLOR_BGR2GRAY as the color space conversion code. We then calculate the images grayscale histograms using the OpenCV "calcHist" function. A grayscale image has only one channel, so we have a value of [0] for channels We don’t have a mask, so we set the mask value to None. We use 256 bins in our histograms, and the possible values range from 0 to 255. The output histograms for the two camera frames are then normalized using the NORM_MINMAX normalization type. Finally, we compute the compare the two normalized grayscale histograms, using the OpenCV "compareHist" function, based on their correlation coefficient, by setting the metric to compute the matching to cv2.HISTCMP_CORREL. If the difference score is greater or equal to our defined threshold, the frames are considered as equal. We return a JSON object with the Boolean of the frame’s similarity and the similarity score.

```
def framesComparison(base64_previous_frame_data, base64_current_frame_data):

    FramesCorrelationThreshold = 0.80

    # Convert previous and current frames to CV images
    previous_frame_image = dataBase64ToCVImage(base64_previous_frame_data)
    current_frame_image = dataBase64ToCVImage(base64_current_frame_data)

    # Convert images to grayscale
    gray_previous_frame_image = cv2.cvtColor(previous_frame_image, cv2.COLOR_BGR2GRAY)
    gray_current_frame_image = cv2.cvtColor(current_frame_image, cv2.COLOR_BGR2GRAY)

    # Calculate the images grayscale histograms
    #  A grayscale image has only one channel, so we have a value of [0] for channels
    #  We don’t have a mask, so we set the mask value to None.
    #  We will use 256 bins in our histogram, and the possible values range from 0 to 255.
    hist_previous_frame = cv2.calcHist([gray_previous_frame_image], [0], None, [256], [0, 256])
    hist_current_frame = cv2.calcHist([gray_current_frame_image], [0], None, [256], [0, 256])


    # normalize the histograms
    cv2.normalize(hist_previous_frame, hist_previous_frame, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX)
    cv2.normalize(hist_current_frame, hist_current_frame, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX)

    # plot the normalized histogram
    # plotHistogram(hist_previous_frame)
    # plotHistogram(hist_current_frame)

    framesHistDiff = cv2.compareHist(hist_previous_frame, hist_current_frame, cv2.HISTCMP_CORREL)

    output = {}
    output['areFramesSimilar'] = framesHistDiff >= FramesCorrelationThreshold
    output['score'] = framesHistDiff

    return output
```

Image 87: Our server script for camera frames similarity computation

8. PERFORMANCE

The main issue for application was not only to make it work but also to create a smooth user experience meaning that it would not bring discomfort to the user to use our application. Also, because we deal with data from our server response it has no use for the user to wait to perform an action for more than some hundred milliseconds. The performance of our device and our applications is controlled by three main threads or basic functionalities, the Application Thread that executes entirely on the CPU, the Render Thread that is initiated on the CPU but gets passed at some point to the GPU and the GPU thread that is controlling the graphics pipelining of any graphical object or component that has to be rendered [90].

Some bottlenecks that anyone may encounter are coming from the Application thread that is running on the CPU of the device. This thread does many things behind the scenes apart from starting or executing our scripts and code.

The next thread, which is the Render Thread, is very fragile to changes that might affect its performance. This thread sends all the requests to the GPU for rendering 3D models or anything we want our application to project to the user. The simplest way to disrupt any execution order for the requests is by simply adding code to the Update() function and accidentally alter the execution order or delay requests to the GPU. The user will immediately experience this case because the application will either start to freeze or miss frames or delay to project any graphical object to the user.

At last, the GPU thread comes in play which executes all the incoming requests from the CPU to render the graphics. In simple words the GPU transforms all the models, UI components etc. to pixels.

Generally, HoloLens applications will be GPU bound, but not always. There are many tools to get a profile of the applications behavior and execution stack. In the next figure we show the lifetime of a frame in our device.

It is obvious that in a single frame many procedures are started and completed. With our applications many times we disrupted the above frame procedures or the order of them. The most visible and immediate consequence was the low frame rate or fps that originally is supported to be at 60fps.

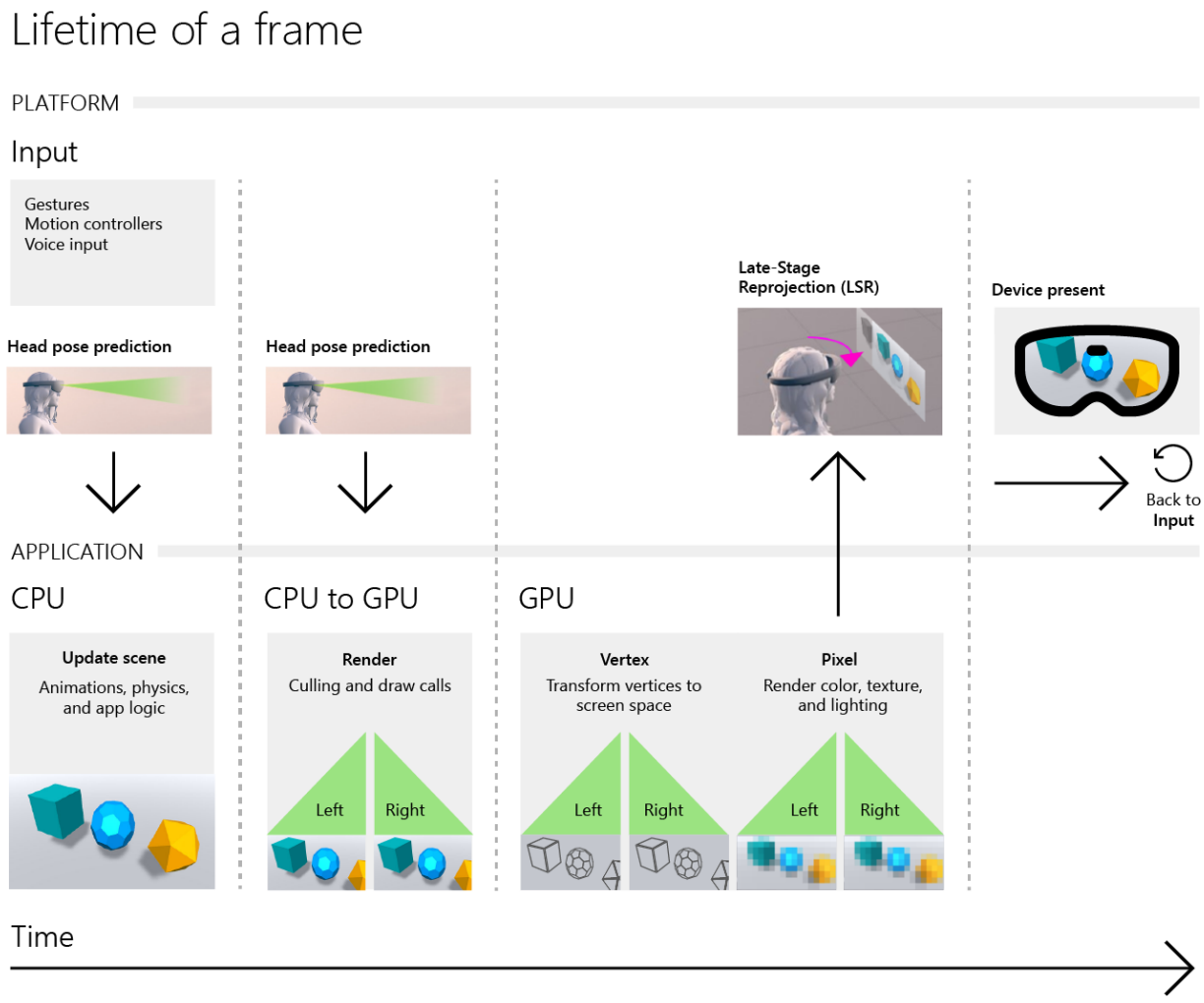


Figure 36: Lifetime of a frame [90]

9. APPLICATION PORTING TO THE NEW HOLOLENS 2

As we described in the previous sections, we used many components that are available on the Mixed Reality Toolkit. Due to the fact that these libraries are created mainly for the HoloLens 2 device we are confident that we will be able to port our application to the new device. In the next section regarding the future work, we discuss this topic.

Because the MRTK was created mainly for the HoloLens 2 device, the only thing required to port our application to the newer device is to change the active configuration profile. We have already tested this change and successfully build the application. We could not deploy it yet due to the lack of a device.

Furthermore, Microsoft offers some support and suggestions along with some instructions on how to proceed to make the changed needed to deploy the applications made for the HoloLens 1 devices.

We also anticipate some changes to the UI because the HoloLens device offers a wider field of view to the user.

10. FUTURE WORK

10.1 Deploying the applications to the new HoloLens 2 device

As we mentioned above, we aim to integrate out application to the new HoloLens 2. In order to do that we have to ensure that our app and libraries will be able to be integrated with the newest version of the MRTK. We are not certain on how easy this is going to be although our application is already set to run on the HoloLens 2. Testing on the device is necessary to guarantee the results.

10.2 Using custom object detection models for various applications

With its modular design, this thesis works as proof-of-concept implementation for the successful integration of Machine Learning on Mixed Reality systems and serves as a blueprint for many future applications. Using a custom object detection model trained for detecting specific classes, this application can be extended to provide an innovative and increasingly accessible path across many fields including design, education, entertainment, military training, healthcare, product content management, and human-in-the-loop operation of robots.

For instance, using a custom object detection model, the application can be extended to detect specific provide audio and visual guidance to users performing high precision manufacturing work, allowing employees to complete repetitive tasks faster and all but eliminating assembly errors. Specifically, car manufacturers could use mixed reality technology to prototype vehicles in a virtual environment. It’s expected that mixed reality applications will help companies with engineering and design modelling, in sales and even with employee training and education and more.

With Microsoft Teams and Dynamics 365 Remote Assist, HoloLens 2 also provides opportunities for hands-free remote assistance. In the medical field, this enables doctors to perform a procedure on a patient, while other specialists provide guidance or analysis through a live video feed from anywhere in the world. An extension of the application for identifying medical-based objects could serve as an additional assistance.

Finally, an extended version of the application could work as a visual prosthesis for the vision impaired, not relaying actual visual data but guiding them in real time with audio cues and instructions.

CONCLUSIONS

To conclude, we presented an application that is not available to any app store or platform that introduces the integration of mixed reality with Machine Learning through the task of object detection.

Since mixed reality maintains a connection to the real world allowing real and virtual elements to interact with one another and the user to interact with virtual elements like they would in the real world, our application takes advantage of these features and superimposes holograms of the detected objects, while the user maintains the connection and interaction with the real world.

We follow the official guidelines for placing our holograms in order to ensure the best user experience. We also utilize the HoloLens device support for voice commands and spatial sound, to make our application accessible to users with visual impairment or blindness, where the object detection results are announced with an audio message.

We believe that this application is going to be the start for a new direction in handling and visualizing object detection data, as a part of the power of Mixed Reality to support the future in many aspects of our lives.

ABBREVIATIONS - ARCTICS - ACRONYMS

MR	Mixed Reality
AR	Augmented Reality
VR	Virtual Reality
DNN	Deep Neural Network
YOLO	You Only Look Once
REST	Representational state transfer
MS COCO	Microsoft Common Objects in Context
API	Application Programming Interface
JSON	JavaScript Object Notation
MRTK	Mixed Reality Toolkit
HMD	Head Mounded Display
FOV	Field of View
ML	Machine Learning
CPU	Central Processing Unit
GPU	Graphical Processing Unit
FPS	Frames per second
UI	User Interface
CNN	Convolutional Neural Network
AP	Average Precision
IoU	Intersection over Union
SSD	Single Shot Detectors
NMS	Non-Maximum Suppression
OID	Open Images Dataset
SLAM	Simultaneous Localization and Mapping
WSGI	Web Server Gateway Interface

APPENDIX I

The source code of this thesis related to the MR HoloLens application can be found by following the link:
https://gitlab.com/mpavlopoulou/hololensobjectdetectionapp
The source code of this thesis related to the server application can be found by following the link:
https://gitlab.com/mpavlopoulou/thesisobjectdetectionproject

Unity Editor can be found here https://unity.com/
Mixed Reality Toolkit: https://github.com/microsoft/MixedRealityToolkit-Unity
Tutorials for HolLens 1 with MRKT:
https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-getting-started

REFERENCES

[1] Mixed Reality Toolkit, https://github.com/microsoft/MixedRealityToolkit-Unity
[2] What is Virtual Reality? Definition and Examples, https://www.marxentlabs.com/what-is-virtualreality/
[3] Demystifying the Virtual Reality Landscape, https://www.intel.com/content/www/us/en/tech-tips-and-tricks/virtual-reality-vs-augmented-reality.html
[4] Virtual, Augmented and Mixed reality explained, https://varjo.com/virtual-augmented-and-mixed-reality-explained/
[5] VR, AR, MR, and What Does Immersion Actually Mean?, https://www.thinkwithgoogle.com/intl/en-cee/future-of-marketing/emerging-technology/vr-ar-mr-and-what-does-immersion-actually-mean/#:~:text=In%20the%20context%20of%20virtual,interact%20with%20the%20virtual%20environment
[6] Abhijit Jana, Mallikarjuna Rao, and Manish Sharma, HoloLens Blueprints, Packt,2017
[7] History of virtual reality: Timeline, https://www.verdict.co.uk/history-virtual-reality-timeline/
[8] The best VR apps in 2022, https://www.creativeblog.com/features/best-vr-apps
[9] Make sense of augmented reality, virtual reality, and mixed reality: Introduction to mixed reality, https://docs.microsoft.com/en-us/learn/modules/intro-to-mixed-reality/3-make-sense
[10] Mission:ISS - Virtual reality application, https://www.oculus.com/experiences/rift/1178419975552187/
[11] How Augmented Reality Works, https://computer.howstuffworks.com/augmented-reality.html
[12] The Difference Between Virtual Reality, Augmented Reality And Mixed Reality https://www.forbes.com/sites/quora/2018/02/02/the-difference-between-virtual-reality-augmented-reality-and-mixed-reality/?sh=6b8732212d07
[13] IKEA to launch new AR capabilities for IKEA Place on new iPad Pro https://about.ikea.com/en/newsroom/2020/03/19/ikea-to-launch-new-ar-capabilities-for-ikea-place-on-new-ipad-pro
[14] NAFTES, https://www.naftes.eu/
[17] Milgram, Paul & Kishino, Fumio. (1994). A Taxonomy of Mixed Reality Visual Displays. IEICE Trans. Information Systems. vol. E77-D, no. 12. 1321-1329.
[18] Mixed Reality documentation: What is mixed reality? https://docs.microsoft.com/en-us/learn/modules/intro-to-mixed-reality/2-what
[19] Microsoft HoloLens: Everything you need to know about the $3,000 AR headsethttps://www.wareable.com/ar/microsoft-hololens-everything-you-need-to-know-about-the-futuristic-ar-headset-735
[20] How to develop for Microsoft HoloLens, https://www.hypergridbusiness.com/2017/01/how-to-develop-for-microsoft-hololens/
[21] P. Milgram and H. Colquhoun Jr., A Taxonomy of Real and Virtual World Display Integration, in Mixed Reality: Merging Real and Virtual Worlds, New York, Springer-Verlag Berlin Heidelberg. (1999) 5-30. Doi: https://doi.org/10.1007/978-3-642-87512-0_1
[22] Mixed Reality documentation: HoloLens 1 Hardware, https://docs.microsoft.com/en-us/hololens/hololens1-hardware
[23] About HoloLens 2, https://docs.microsoft.com/en-us/hololens/hololens2-hardware
[24] Jacek Kościesza, “HoloLens 2 vs HoloLens 1: what’s new?”, https://4experience.co/hololens-2-vs-hololens-1-whats-new/
[25] Unity Editor, https://unity.com/products/core-platform
[26] Mixed Reality documentation: Upgrading from HoloToolkit, https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/updates-deployment/htk-to-mrtk-porting-guide?view=mrtkunity-2022-05
[27] Using the visual profiler, https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/features/diagnostics/using-visual-profiler?view=mrtkunity-2022-05
[28] Mixed Reality documentation: MRTK packages, https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/packages/mrtk-packages?view=mrtkunity-2022-05
[29] NVIDIA Jetson TX2 Developer Kit Specifications, https://developer.nvidia.com/embedded/jetson-tx2-developer-kit
[30] Nvidia Jetpack SDK, https://developer.nvidia.com/embedded/jetpack
[31] Nvidia NGC Containers catalog, https://catalog.ngc.nvidia.com/containers
[32] Nvidia L4T-ML Docker container, https://catalog.ngc.nvidia.com/orgs/nvidia/containers/l4t-ml
[33] Docker platform, https://www.docker.com/
[34] Flask Python library, https://flask.palletsprojects.com/en/2.1.x/
[35] OpenCV library, https://opencv.org/
[36] Mixed Reality documentation: What is a hologram, https://docs.microsoft.com/en-us/windows/mixed-reality/discover/hologram
[37] How to develop for Microsoft HoloLens, https://www.hypergridbusiness.com/2017/01/how-to-develop-for-microsoft-hololens/

[38] Mixed Reality documentation: Hologram stability, https://docs.microsoft.com/en-us/windows/mixed-reality/develop/advanced-concepts/hologram-stability
[39] Mixed Reality documentation: Coordinate systems, https://docs.microsoft.com/en-us/windows/mixed-reality/design/coordinate-systems
[40] Mixed Reality documentation: Gaze, https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/features/input/gaze?view=mrtkunity-2021-05
[41] Mixed Reality documentation: Gestures, https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/features/input/gestures?view=mrtkunity-2021-05
[42] Mixed Reality documentation: Getting around HoloLens (1st gen), https://docs.microsoft.com/en-us/hololens/hololens1-basic-usage
[43] Klint, L. 2018. HoloLens Succinctly. E-book. Syncfusion, Inc. Available at: https://www.syncfusion.com.
[44] Mixed Reality documentation: Voice input, https://docs.microsoft.com/en-us/windows/mixed-reality/design/voice-input
[45] Mixed Reality documentation: Spatial mapping, https://docs.microsoft.com/en-us/windows/mixed-reality/design/spatial-mapping
[46] Mixed Reality documentation: Spatial anchors, https://docs.microsoft.com/en-us/windows/mixed-reality/design/spatial-anchors
[47] Mixed Reality documentation: Spatial sound overview, https://docs.microsoft.com/en-us/windows/mixed-reality/design/spatial-sound
[48] Mixed Reality documentation: Follow toggle, https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/features/ux-building-blocks/solvers/solver?view=mrtkunity-2022-05#follow
[49] Object detection guide, https://www.fritz.ai/object-detection/
[50] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” arXiv:1506.02640v5, 2016. Available: https://arxiv.org/abs/1506.02640v5
[51] Howard, Andrew & Zhu, Menglong & Chen, Bo & Kalenichenko, Dmitry & Wang, Weijun & Weyand, Tobias & Andreetto, Marco & Adam, Hartwig. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.
[52] Wu, Bichen & Iandola, Forrest & Jin, Peter & Keutzer, Kurt. (2017). SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving. 446-454. 10.1109/CVPRW.2017.60.
[53] YOLO: Real-Time Object Detection, https://pjreddie.com/darknet/yolo/
[54] Digging deep into YOLO V3, https://towardsdatascience.com/digging-deep-into-yolo-v3-a-hands-on-guide-part-1-78681f2c7e29
[55] YOLOv3: Real-Time Object Detection Algorithm (What’s New?), https://viso.ai/deep-learning/yolov3-overview/#:~:text=YOLOv3%20AI%20models-,What%20is%20YOLOv3%3F,network%20to%20detect%20an%20object.
[56] Wikipedia - Object detection, https://en.wikipedia.org/wiki/Object_detection#:~:text=Object%20detection%20is%20a%20computer,in%20digital%20images%20and%20videos.
[57] Ghenescu, Veta & Mihaescu, Roxana & Carata, Serban-Vasile & Ghenescu, Marian & Barnoviciu, Eduard & Chindea, Mihai. (2018). Face Detection and Recognition Based on General Purpose DNN Object Detector. 1-4. 10.1109/ISETC.2018.8583861.
[58] Redmon, Joseph & Farhadi, Ali. (2018). YOLOv3: An Incremental Improvement.
[59] What’s new in YOLO v3, https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b
[60] The beginner’s guide to implementing YOLOv3 in TensorFlow 2.0, https://machinelearningspace.com/yolov3-tensorflow-2-part-1/#nms-unique
[61] Introduction to the COCO Dataset, https://opencv.org/introduction-to-the-coco-dataset/
[62] MS COCO Dataset, https://cocodataset.org/#home
[63] Set up a mixed reality project in Unity with the Mixed Reality Toolkit https://docs.microsoft.com/en-us/learn/modules/mixed-reality-toolkit-project-unity/
[64] Mixed Reality documentation: Install the tools, https://docs.microsoft.com/el-gr/windows/mixed-reality/develop/install-the-tools
[65] Unity documentation: Coroutines, https://docs.unity3d.com/Manual/Coroutines.html
[66] Mixed Reality documentation: Buttons, https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/features/ux-building-blocks/button?view=mrtkunity-2022-05#button-prefabs-in-mrtk
[67] Mixed Reality documentation: Examples Hub, https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/running-examples-hub?view=mrtkunity-2022-05
[68] Mixed Reality documentation: Handling speech input, https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/features/input/speech?view=mrtkunity-2022-05#handling-speech-input

[69] Mixed Reality documentation: TextToSpeech Class, https://docs.microsoft.com/en-us/dotnet/api/microsoft.mixedreality.toolkit.audio.texttospeech?view=mixed-reality-toolkit-unity-2020-dotnet-2.8.0
[70] Mixed Reality documentation: Tooltip, https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/features/ux-building-blocks/tooltip?view=mrtkunity-2022-05
[71] Unity documentation: Spatial Mapping components, https://docs.unity3d.com/560/Documentation/Manual/windowsholographic-sm-component.html
[72] Unity documentation: Spatial Mapping Collider, https://docs.unity3d.com/2019.1/Documentation/Manual/SpatialMappingCollider.html
[73] Mixed Reality documentation: Head-gaze in Unity, https://docs.microsoft.com/en-us/windows/mixed-reality/develop/unity/gaze-in-unity
[74] Unity documentation: Line Renderer component, https://docs.unity3d.com/Manual/class-LineRenderer.html
[75] Unity documentation: AudioSource, https://docs.unity3d.com/ScriptReference/AudioSource.html
[76] Unity documentation: AudioClip, https://docs.unity3d.com/ScriptReference/AudioClip.html
[77] Mixed Reality documentation: Holographic Rendering overview, https://docs.microsoft.com/en-us/windows/mixed-reality/develop/advanced-concepts/rendering-overview
[78] Unity documentation: HoloLens photo capture, https://docs.unity3d.com/2018.2/Documentation/Manual/windowsholographic-photocapture.html
[79] Mixed Reality documentation: Locatable camera, https://github.com/MicrosoftDocs/mixed-reality/blob/a0b2c53dc295db0332832ba00b60bd7a4962e3d9/mixed-reality-docs/locatable-camera.md
[80] Gu, Fuqiang & Hu, Xuke & Ramezani, Milad & Acharya, Debaditya & Khoshelham, Kourosh & Valaee, Shahrokh & Shang, Jianga. (2019). Indoor Localization Improved by Spatial Context - A Survey. ACM Computing Surveys. 52. 64:1-35. 10.1145/3322241.
[81] Nießner, Matthias & Zollhöfer, Michael & Izadi, Shahram & Stamminger, Marc. (2013). Real-time 3D Reconstruction at Scale using Voxel Hashing. ACM Transactions on Graphics (TOG). 32. 10.1145/2508363.2508374.
[82] B. Glocker, J. Shotton, A. Criminisi and S. Izadi, "Real-Time RGB-D Camera Relocalization via Randomized Ferns for Keyframe Encoding," in IEEE Transactions on Visualization and Computer Graphics, vol. 21, no. 5, pp. 571-583, 1 May 2015, doi: 10.1109/TVCG.2014.2360403.
[83] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," in IEEE Transactions on Robotics, vol. 33, no. 5, pp. 1255-1262, Oct. 2017, doi: 10.1109/TRO.2017.2705103.
[84] Tola, E., 2005. Multiview 3D Reconstruction of a scene containing independently moving objects (Master's thesis).
[85] Poothicottu Jacob, G.: HoloLens framework for augmented reality applications in breast cancer surgery. Master's thesis, University of Waterloo (2018)
[86] Labini, Mauro & Gsaxner, Christina & Pepe, Antonio & Wallner, Jürgen & Egger, Jan & Bevilacqua, Vitoantonio. (2019). Depth-Awareness in a System for Mixed-Reality Aided Surgical Procedures. 10.1007/978-3-030-26766-7_65
[87] Microsoft: Process Media frames with MediaFrameReader, https://docs.microsoft.com/enus/windows/uwp/audio-video-camera/processmedia-frames-with-mediaframereader#settingup-your-project
[88] Fan, J. I. and Khoshelham, K.: Augmented Reality Asset tracking using HoloLens, ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci., V-4-2021, 121–127, https://doi.org/10.5194/isprs-annals-V-4-2021-121-2021, 2021.
[89] Farasin, Alessandro & Peciarolo, Francesco & Grangetto, Marco & Gianaria, Elena & Garza, Paolo. (2020). Real-time Object Detection and Tracking in Mixed Reality using Microsoft HoloLens. 165-172. 10.5220/0008877901650172.
[90] Mixed Reality documentation: Understanding performance for mixed reality, https://docs.microsoft.com/en-us/windows/mixed-reality/develop/advanced-concepts/understanding-performance-for-mixed-reality