

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Mixed Reality System for On-the-Job Training on Industrial Operations

Ana Ricardo Pinto



Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Vítor Hugo Pinto

November 3, 2025

Resumo

O aumento da complexidade das tarefas industriais, impulsionado pelos avanços tecnológicos, tem evidenciado as limitações dos métodos tradicionais de formação de operadores, nomeadamente os elevados custos, a falta de experiência prática e as ineficiências na gestão do tempo. Para enfrentar estes desafios, esta dissertação explora a utilização de tecnologias de Realidade Mista para melhorar a formação de operadores em tarefas de montagem industrial. Ao combinar hologramas espacialmente alinhados, feedback em tempo real e interação sem mãos, o sistema proposto visa reduzir o tempo de formação, aumentar a eficácia da aprendizagem e minimizar os riscos físicos para os operadores recém-contratados.

Foi desenvolvida uma aplicação personalizada de realidade mista utilizando o Unity, o Vuforia e o Microsoft MRTK, que orienta os utilizadores passo a passo no processo de montagem de um espelho lateral numa porta de carrinha.

Os resultados demonstram a viabilidade e adaptabilidade do sistema, mas também revelam limitações atuais, como restrições na deteção de um único alvo e ocasionais imprecisões na deteção de objetos. O feedback é fornecido através de mensagens de texto, oferecendo uma orientação básica mas eficaz, com potencial para ser expandida para modalidades de interação mais ricas. No geral, este trabalho apresenta uma solução de formação flexível e escalável, alinhada com o crescente interesse pelas tecnologias imersivas no setor da manufatura e na educação técnica.

Abstract

The growing complexity of industrial tasks in the context of technological advancements has highlighted the limitations of traditional operator training methods, such as high costs, insufficient hands-on experience, and inefficiencies in time management. To address these challenges, this dissertation explores the use of Mixed Reality technologies to improve operator training in industrial assembly tasks. By combining spatially aligned holograms, real-time feedback, and hands-free interaction, the proposed system aims to reduce training time, enhance learning effectiveness, and minimize physical risks for newly hired operators. A custom mixed reality application was developed using Unity, Vuforia, and Microsoft's MRTK, guiding users through the step-by-step assembly of a wing mirror onto a van door.

Results demonstrate the feasibility and adaptability of the system, but also reveal current limitations, such as single-target tracking constraints and occasional inaccuracies in object detection. Feedback is provided via text prompts, offering basic but effective guidance, with potential for expansion to richer interaction modalities. Overall, this work presents a flexible and scalable training solution that aligns with the growing interest in immersive technologies within manufacturing and technical education.

UN Sustainable Development Goals

The United Nations Sustainable Development Goals (SDGs) provide a global framework to achieve a more sustainable and equitable future. They consist of 17 goals addressing the world's major challenges, including poverty, inequality, climate change, and innovation.

Although primarily focused on social and environmental issues, several SDGs are directly relevant to this work. By improving operator skills, reducing training inefficiencies, and minimizing workplace risks, this dissertation contributes to a more inclusive and technologically advanced industrial sector.

The specific Sustainable Development Goals considered are:

SDG 4 Ensure inclusive and equitable quality education and promote lifelong learning opportunities for all.

SDG 8 Promote sustained, inclusive and sustainable economic growth, full and productive employment and decent work for all.

SDG 9 Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation.

SDG	Target	Contribution	Performance Indicators and Metrics
4	4.3	MR training provides immersive, step-by-step guidance in industrial tasks, improving access to quality vocational education.	Operators trained; completion rate; skill acquisition
	4.4	MR-based learning enhances employability through relevant technical competencies.	Proficiency rate post-training
8	8.5	Reducing training time and errors increases productivity and promotes safer working conditions.	Training duration; error rate; safety incidents
9	9.2	MR integration fosters innovation in industrial processes and supports sustainable industrialization.	MR modules deployed; adoption rate
	9.5	MR-based solutions strengthen research and technological capabilities in industry.	Research outputs; technological readiness level achieved

Agradecimentos

Embora a elaboração de uma dissertação possa ser um processo solitário e individual, há sempre pessoas que nos apoiam e ajudam ao longo do caminho.

Quero agradecer ao Pedro Mota, pela ajuda — sem qualquer obrigação — em algumas partes deste trabalho e pela preocupação.

Também ao meu orientador, Vítor Pinto, pelo apoio e confiança, e ao Lucas Rocha, alguém a quem podia sempre recorrer para esclarecer dúvidas específicas sobre as ferramentas utilizadas.

À minha mãe, que fez questão de ajudar e rever parte do texto. Ao meu tio que me emprestou o computador quando fiquei sem PC duas vezes durante o desenvolvimento deste trabalho.

E, claro, não menos importante, agradeço ainda a todas as pessoas que, de uma forma ou de outra, contribuíram para a conclusão desta etapa — seja ouvindo desabafos, oferecendo conselhos ou com um abraço reconfortante.

Obrigada a todos; sozinha, não estaria aqui.

Ana Ricardo Pinto

“So long, goodbye. Do I really have to finish?”

Bo Burnham

Contents

Acknowledgements	vii
1 Introduction	1
1.1 Scientific Contribution	1
1.2 Document Structure	2
2 Preliminary Study	3
2.1 Background and Fundamental Aspects	3
2.1.1 Mixed-Reality	3
2.1.2 Digital Twins	4
2.1.3 Effectiveness and relevance of using MR in training	5
2.2 Related Work	6
2.2.1 Assembly Task Training	6
2.2.2 Assembly Task Guidance	8
2.2.3 Human-Robot Collaboration	12
2.3 Visualization Mediums	16
2.3.1 HMDs	16
2.3.2 Mobile Devices	17
2.3.3 Projectors	17
2.3.4 Summary of Visualization Mediums	17
2.4 Software	19
2.4.1 Engines/Simulators	19
2.4.2 Platforms	20
2.4.3 Ready-to-use MR Applications	20
2.4.4 Summary of Software	21
3 Practical Implementation	23
3.1 Use Case	23
3.1.1 Material	23
3.1.2 Assembly Steps	25
3.2 Development	27
3.2.1 Tools	27
3.2.2 3D Models	29
3.2.3 Object Tracking	30
3.2.4 Training Modes	31
3.2.5 Main Scene	32
3.2.6 Application Deployment	41
3.2.7 Version Control and Project Repository	43

4	Experimental Results and Discussion	45
4.1	Simulation in Unity	45
4.1.1	Unity's Play Mode	45
4.1.2	Start Menu	46
4.1.3	Main Scene	47
4.2	HoloLens Deployment	54
4.2.1	Start Menu	54
4.2.2	Main Scene	55
4.2.3	Limitations and Challenges	63
4.3	Discussion	66
5	Conclusion and Future Work	71
5.1	Conclusions	71
5.2	Future Work	72
	References	75
A	Unity Project Scripts	79
A.1	Mode Manager Script	79
A.2	Assembly Logic Scripts	80
A.3	Feedback Helper Scripts	104

List of Figures

2.1	Virtuality Continuum defined by Milgram and Kishino [3].	3
2.2	Data flow in different Digital Twin categories: (a) Digital Model, (b) Digital Shadow, and (c) Digital Twin, taken from [7].	5
2.3	Instructions given in the training modes developed by Apostolopoulos et al. [12] for operators: (a) Mode 1 and (b) Mode 2.	7
2.4	AR visualization of task information in real space (A), and smartwatch interface for robot guidance control (B) — taken from Michalos et al. [17].	9
2.5	Mixed Reality interface for assembly support — adapted from Fathi et al. [21].	11
2.6	Working environment of the approach proposed by Choi et al. [23] for safety-aware human-robot interaction.	13
2.7	Microsoft HoloLens 2 headset	16
3.1	Van door (without the quarter glass piece installed)	24
3.2	Wing mirror	24
3.3	Van door with quarter glass piece installed	25
3.4	Correct cable position, passed through the attachment piece on the quarter glass.	26
3.5	Mirror correctly placed onto door	26
3.6	Screw hole location (seen from back of door)	27
3.7	Door 3D model	29
3.8	Wing mirror 3D model	29
3.9	Quarter-glass window attachment piece 3D model	30
3.10	UI element used in the Start Menu to select the training mode.	32
3.11	Hierarchy of the Main Scene, highlighting the primary GameObjects used in the MR application.	33
3.12	MRTK GameObjects in the Main Scene, responsible for mixed reality interactions and UI elements.	34
3.13	UML State Machine Diagram showcasing the revised behavior and states of the main scene of the MR application.	35
3.14	"Ghost" counterpart of the wing mirror model used as a placement guide.	37
3.15	Dialog prompt asking the user to confirm they have understood the mirror placement instructions.	37
3.16	Dialog prompt asking the user to confirm they have placed the mirror.	38
3.17	Screw hologram and visual guide object for each screw hole.	39
3.18	Unity build profiles window	42
3.19	Deployment process in Visual Studio — with the top bar highlighted in red	43
4.1	Mirror Assembly Phase in Unity's Play Mode with different training modes: (a) Virtual Mode, (b) Mixed Mode, and (c) Full Mode.	46

4.2	Enabling Model Target Visualization in Vuforia's configuration settings within Unity.	46
4.3	Start Menu scene within Unity's Play Mode	47
4.4	Prompting the user to look at the door for detection in Mixed Mode (and Full Mode) within Unity's Play Mode.	47
4.5	Real-time feedback in Unity's Play Mode: the system displays a ghost mirror and on-screen text to guide correct placement.	48
4.6	Text feedback in Unity's Play Mode for correct mirror placement	49
4.7	Text feedback in Unity's Play Mode to insert the screws.	49
4.8	Text feedback in Unity's Play Mode when one out of three screws is inserted. . .	50
4.9	Text feedback in Unity's Play Mode when two out of three screws are inserted. .	50
4.10	"Assembly Complete" text shown in Unity's Play Mode once assembly is complete.	51
4.11	Mirror ghost animation in Full Mode within Unity's Play Mode: (a) initial position, and (b) final position on the door.	51
4.12	Dialog for the user to dismiss the mirror ghost animation in Full Mode within Unity's Play Mode.	52
4.13	User confirming they have placed the mirror via the dialog in Full Mode within Unity's Play Mode.	52
4.14	Prompting the user to look at the mirror for detection in Full Mode within Unity's Play Mode.	53
4.15	Text Feedback in Unity's Play Mode for incorrect mirror placement in Full Mode.	53
4.16	Text Feedback in Unity's Play Mode for correct mirror placement in Full Mode. .	54
4.17	Start menu element in HoloLens deployment	55
4.18	Virtual elements placed onto the real world in Virtual Mode.	55
4.19	Mirror placement guided by ghost hologram and text feedback in Virtual Mode. .	56
4.20	Prompt in Virtual Mode guiding the user to insert holographic screws in HoloLens deployment.	56
4.21	Text Feedback in Virtual Mode when screws are inserted in HoloLens deployment: (a) first screw, (b) second screw, and (c) all screws inserted.	57
4.22	"Assembly Complete" text shown in Virtual Mode once assembly is complete in HoloLens deployment.	57
4.23	Prompting the user to look at the door for detection in Mixed Mode.	58
4.24	Mirror placement guided by ghost hologram and text feedback in Mixed Mode. .	58
4.25	Text feedback in Mixed Mode for correct mirror placement in HoloLens deployment.	59
4.26	Prompt in Mixed Mode guiding the user to insert holographic screws in HoloLens deployment.	59
4.27	Text feedback in Mixed Mode when screws are inserted in HoloLens deployment: (a) first screw, (b) second screw, and (c) all screws inserted.	59
4.28	"Assembly Complete" text shown in Mixed Mode once assembly is complete in HoloLens deployment.	60
4.29	Mirror ghost animation in Full Mode within HoloLens deployment: (a) initial position, and (b) final position on the door.	61
4.30	Dialog for the user to dismiss the mirror ghost animation in Full Mode within HoloLens deployment.	61
4.31	User confirming they have placed the mirror via dialog in Full Mode within HoloLens deployment.	61
4.32	Prompting the user to look at the mirror for detection in Full Mode within HoloLens deployment.	62

4.33	Mirror placement feedback in Full Mode within HoloLens deployment: (a) incorrect placement, and (b) correct placement.	62
4.34	Prompt in Full Mode guiding the user to insert holographic screws in HoloLens deployment.	62
4.35	Text feedback in Full Mode when screws are inserted in HoloLens deployment: (a) first screw, (b) second screw, and (c) all screws inserted.	63
4.36	Example of misalignment: the mirror ghost hologram appears offset from the actual mirror position in Full Mode.	64
4.37	Example of misalignment: the holographic screws do not perfectly align with the physical holes in Full Mode.	64
4.38	Comparison between the real van door (a) and its 3D model (b), highlighting differences around the screw holes.	65

List of Tables

2.1	Summary of Visualization Technologies used in reviewed papers	18
2.2	Summary of Software used in reviewed papers	21
3.1	Software tools and versions used in development	28
3.2	Components status' in different training modes	31
4.1	Comparison between studies and the present work.	67

List of Acronyms

AGV	Automated Guided Vehicle
AI	Artificial Intelligence
AR	Augmented Reality
CAD	Computer-Aided Design
DT	Digital Twin
HMD	Head-Mounted Display
HRC	Human-Robot Collaboration
MR	Mixed Reality
MRTK	Mixed Reality Toolkit
PLC	Programmable Logic Controller
UI	User Interface
VR	Virtual Reality

Chapter 1

Introduction

The rise of Industry 4.0 has ushered in a transformative era of technological innovation, marked by advancements in autonomous systems, the Internet of Things, and immersive technologies. This era represents a fusion of technologies across the physical and digital domains, fundamentally altering how we live, work, and interact [1]. These advancements are crucial not only for optimizing production processes but also for improving human-machine collaboration, allowing for more efficient and safer work environments.

Despite this progress, training workers on complex industrial tasks still often requires considerable resources, consuming both time and financial investment. Traditional methods often lack the hands-on experience required for effective learning, resulting in extended training periods, inconsistent skill acquisition, and increased operational costs due to inefficiencies and human error.

In parallel, over the past decade, the trajectory of Industry 4.0 has leaned heavily toward digitalization and AI-driven productivity—often at the expense of its original principles of social responsibility and sustainability. In response, the concept of Industry 5.0 has emerged, placing renewed emphasis on human-centric, resilient, and sustainable innovation. It promotes the development of technologies that adapt to the worker, rather than forcing the worker to adapt to the technology, and supports the use of digital tools to increase flexibility, foster circularity, and deliver societal value [2].

Within this evolving landscape, Mixed Reality (MR) and Digital Twin (DT) technologies offer a promising path forward. By enabling immersive, contextualized, and interactive learning, these tools can enhance training while reducing physical risk and material waste. Most importantly, they align with the human-centered vision of Industry 5.0 by combining the strengths of both human intuition and machine precision. Leveraging MR for operator support could not only improve technical proficiency but also place the worker at the center of the digital transformation.

1.1 Scientific Contribution

The primary goal of this dissertation is to investigate how mixed reality technologies can be used to improve the training of newly hired operators in industrial environments. Specifically, this

work proposes a MR-based training system that enables immersive, hands-free learning through spatially aligned virtual guidance over physical components.

The main scientific contribution of this dissertation is the design and implementation of a custom MR training application using a game engine, enabling immersive step-by-step guidance for complex assembly tasks. Unlike commercial solutions, this approach allows full control over the user interface, interaction design, and tracking logic, increasing adaptability to specific industrial needs.

1.2 Document Structure

In addition to the introduction, this dissertation is composed of four main chapters.

Chapter 2 presents the technical background of this work, covering the theoretical foundations, related concepts, and current research developments relevant to Mixed Reality and industrial training.

Chapter 3 describes the use case developed for this dissertation, detailing the design and implementation process of the proposed MR training system.

Chapter 4 outlines the results obtained, illustrating the system's behavior across different training modes and discussing the main observations.

Finally, Chapter 5 summarizes the key conclusions drawn from the project, highlights the system's contributions, and suggests directions for future work.

Chapter 2

Preliminary Study

This chapter presents the preliminary study conducted to support the development of the proposed solution. It begins by outlining the key theoretical concepts relevant to this research, with the aim of clarifying the terminology and definitions adopted throughout the dissertation.

In addition to the theoretical background, this chapter provides a review of the current state of the art in MR/AR and DT applications, focusing on their roles in improving human-robot interaction, task training and assistance, and fault detection. The chapter also examines common visualization methods and software tools, and discusses their presence and relevance in the reviewed literature.

2.1 Background and Fundamental Aspects

This dissertation utilizes a set of key concepts and technologies in order to improve the training of operators. This section provides an overview of the most relevant foundational elements, offering context for the technological approach adopted in this work.

2.1.1 Mixed-Reality

One of the key aspects of the dissertation is the use of Mixed Reality. The term Mixed Reality was coined and popularized by Milgram and Kishino [3]. In this article, they define a "Virtuality Continuum", shown in Figure 2.1.

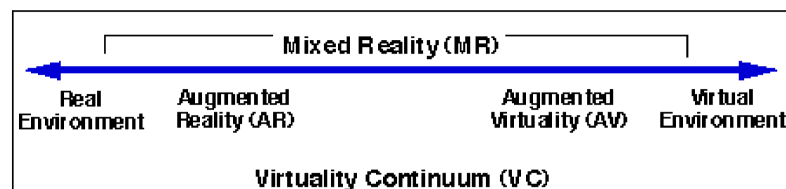


Figure 2.1: Virtuality Continuum defined by Milgram and Kishino [3].

The continuum illustrates the spectrum of environments that combine real and virtual elements. At one end lies the real environment, consisting entirely of real-world elements, such as a

recording of a physical scene. At the opposite end is the virtual environment, where all elements are digital, such as in a fully immersive Virtual Reality (VR) game.

Mixed Reality occupies the intermediate space on this spectrum, incorporating elements of both the real and virtual worlds. According to Milgram and Kishino's definition, any environment blending real and virtual components can be classified as MR, including Augmented Reality (AR) as a subset.

However, in more recent interpretations, some researchers [4] [5] have begun to use the term Mixed Reality specifically to describe cases where there is a two-way interaction between the real and virtual environments. In this context, virtual objects can interact with and respond to physical objects, and actions in one environment can influence the other. For instance, MR may allow users to manipulate a virtual object with their real hands or make digital elements "adhere" to physical surfaces in a way that reacts as if both were physical objects.

This interpretation aims to distinguish MR from AR, where digital information is primarily overlaid onto the real world without significant interaction between virtual and physical elements. MR, in contrast, emphasizes a deeper integration, where digital and physical components can dynamically influence each other. However, the boundaries between these definitions can be difficult to define, and there is ongoing discussion regarding the appropriate terminology and classifications [6].

This dissertation adopts the interpretation of MR that emphasizes its ability to blend virtual components with physical surroundings, making it particularly valuable in training environments.

2.1.2 Digital Twins

Digital Twin technology is another foundational aspect of this dissertation, offering a powerful tool for enhancing industrial training through realistic simulations. A Digital Twin is often described as simply a virtual replica of a physical asset. However, a further distinction can be made between subcategories of Digital Twins: Digital Twins, Digital Shadows, and Digital Models. These terms differ in the level of data integration between the physical and digital counterparts [7].

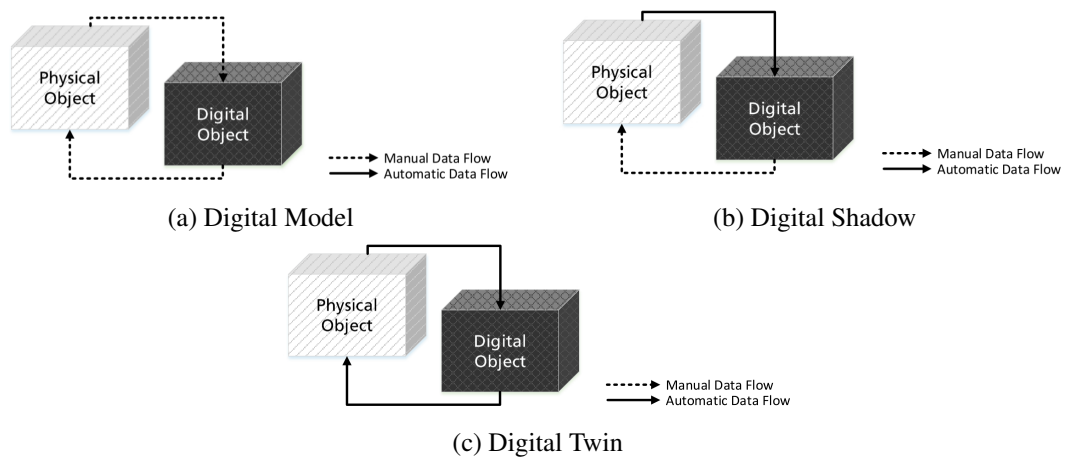


Figure 2.2: Data flow in different Digital Twin categories: (a) Digital Model, (b) Digital Shadow, and (c) Digital Twin, taken from [7].

As illustrated in Figure 2.2, a Digital Model (2.2a) lacks automated data flow between the two objects; an example of this could be a simulation model of a factory. A Digital Shadow (2.2b), on the other hand, features a one-way automated flow of data from the physical object to the digital counterpart. In this scenario, any change in the physical object is reflected in the digital object, although the reverse does not occur. In contrast, in a Digital Twin (2.2c), there is automated data flow between both objects, meaning that a state change in one object directly results in a corresponding change in the other. This bidirectional integration allows the digital model to be used as a way to control the physical object [7].

Digital Models and Digital Shadows represent two approaches to simulating real-world scenarios without the risks associated with physical alterations. Both can be utilized to simulate an assembly workstation to train an operator effectively in a controlled environment. While a Digital Model would provide a simulated representation of the physical station, a Digital Shadow could allow operators to observe and react to real-time data.

In contrast, Digital Twins provide a more interactive and dynamic experience. With a Digital Twin, real-time feedback and control are enabled. This means that operators cannot only simulate tasks and make adjustments in the digital environment but also directly influence the physical object. This interactivity allows for a deeper and more comprehensive understanding of complex operations, but also carries more risk.

Ultimately, if the primary goal is to simulate the workstation, a Digital Model or even a Digital Shadow may be more suitable. However, as operators' skills advance, a full Digital Twin could further enhance their capabilities by enabling real-time interaction with the physical system.

2.1.3 Effectiveness and relevance of using MR in training

Mixed Reality holds significant potential for training operators in assembly tasks by harnessing its advanced capabilities to enhance vocational skills and boost engagement.

A systematic review by Chiang et al. [8] on the use of AR in vocational training highlights several key benefits, including improvements in assembly skills, spatial cognition and visualization, decision-making, and hazard identification. The study also found that AR can reduce cognitive load and shorten training time, leading to overall improvements in training outcomes.

Furthermore, the authors observed a growing interest in the topic over the years—a trend also confirmed by Han et al. [9], who conducted a meta-analysis on the effectiveness of AR-based training. While the reported effect size was modest, it was more pronounced in engineering and technical education compared to other disciplines, indicating greater suitability in these fields.

However, it is important to note that much of the literature reviewed in these studies is based on earlier generations of AR technology. Since then, significant advancements have occurred—notably, the clearer distinction between AR and MR, and a broader focus on increasing interactivity and immersion. These developments suggest that modern MR solutions could provide even more impactful training experiences by offering deeper engagement and more natural interaction, potentially leading to further gains in skill acquisition and performance.

These advantages position MR as a valuable tool for organizations aiming to modernize their training methodologies. A study from 2022 [10] reflects this growing interest: while many companies are already experimenting with or implementing immersive technologies such as AR and MR, others are planning to adopt them in the near future.

However, the same study also notes that a significant number of companies have no current plans for adoption. Reported concerns include high costs, technological complexity, implementation time, and a lack of clear, practical applications for their specific use cases.

While MR and AR technologies show considerable promise for enhancing worker training and support, these limitations must not be overlooked. Continued research and development are necessary to address these barriers, improve accessibility, and enable wider adoption across industrial sectors.

2.2 Related Work

To provide a structured review of the state of the art, this section categorizes the literature into themes based on their contributions to task training, task guidance, human-robot collaboration, and fault detection.

2.2.1 Assembly Task Training

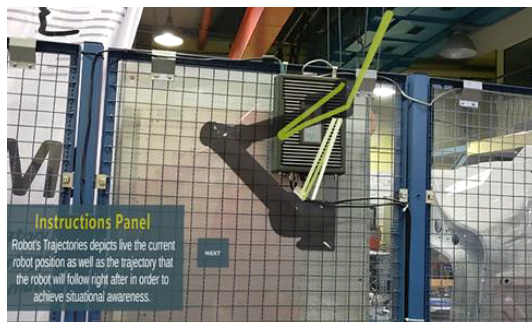
As discussed, AR/MR can play an important role in operator training.

Studies like Sarhan et al. [11] evaluated the effectiveness of AR in supporting knowledge acquisition and transfer in industrial training scenarios. Using an off-the-shelf AR instruction creation platform, they compared AR-based instructions with traditional paper-based methods. Results showed improved task comprehension and performance with AR: participants using AR instructions made approximately half as many errors as those using paper-based guidance.

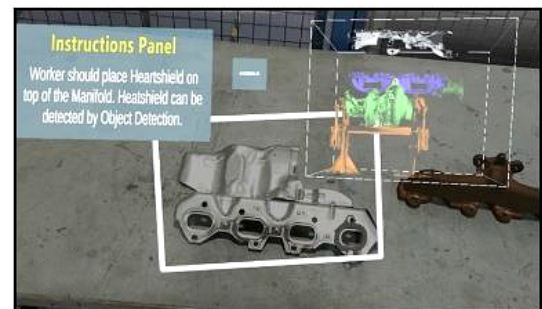
However, the study highlighted several limitations of the AR authoring tool, including reliance on a laptop, lack of voice commands, and restrictions on media types and 3D models per step. To overcome these issues, the authors developed a more flexible solution using a game engine, allowing better interface design, 3D interaction, and eye-tracking integration—while reducing dependency on external hardware. This conclusion shows the importance of customized solutions to fully leverage AR’s potential in training applications.

Apostolopoulos et al. [12] propose an innovative operator training system featuring two distinct modes. Mode 1 focuses on familiarizing operators with hybrid workstations by walking them through the available functionalities and resources. Operators can navigate the environment either physically or remotely while receiving augmented instructions about key elements such as tools, controls, and critical production events (e.g., safety incidents or customized assembly requirements). For example, Figure 2.3a illustrates the introduction of robotic resources and trajectory visualizations.

Figure 2.3b illustrates Mode 2, showcasing an assembly step with augmented visual cues and real-time object tracking within the physical environment. Mode 2 supports product assembly training through a step-by-step AR walkthrough, featuring textual instructions, 3D visualizations, directional arrows, animations, and object identification.



(a) Mode 1 — Teaching robot resources and visualizing trajectories



(b) Mode 2 — Tracking objects and representing assembly steps with augmented components

Figure 2.3: Instructions given in the training modes developed by Apostolopoulos et al. [12] for operators: (a) Mode 1 and (b) Mode 2.

For both operational modes, augmented reality instructions are spatially anchored to physical components tracked in real-time using OpenCV (an open-source computer vision library). While this tracking system is efficient, it operates via 2D object detection. This fundamental limitation restricts the application to a basic superimposition of information onto the real environment.

Nonetheless, the system developed in this study demonstrated high usability and effectiveness for both experienced and novice users, underscoring the potential of AR-based operator training to be intuitive and accessible. By providing context-sensitive information and enabling hands-on training within an augmented physical environment, AR reduced cognitive load and improved users’ comprehension of collaborative systems.

In a project similar to this dissertation and conducted in the same laboratory, Barros [13] focuses on instructing on the assembly of a vehicle door panel using a mixed reality headset combined with a ready-to-use MR application for step-by-step assembly instructions.

This approach employs a QR code anchor: a printed code attached to the physical object (the door) which, when scanned by the MR device, enables recognition and precise alignment of virtual content with the physical environment. This ensures that the MR experience is accurately anchored to the real-world position and orientation of the object.

However, this method faces limitations in realistic industrial environments. It relies on the visibility of the QR code anchor, which reduces flexibility—for instance, the code may not be visible when the user views the door from the opposite side. Furthermore, using this approach in assembly lines would be impractical, as attaching QR codes to every component poses significant challenges for scalability and adaptability. While effective in controlled settings and suitable for simple training applications, its practicality in more complex or real-time scenarios remains limited.

Still regarding operator training, a recent paper [14] describes the development of a training system created in collaboration with a Swedish company to assist operators in manual assembly tasks. The researchers introduced a flexible solution that can transition between different levels of mixing real and virtual elements based on specific requirements. This approach allows users to customize the training environment, ranging from a fully virtual scenario, with only digital assets, to a full production environment, where the system supports operators in assembling physical products in actual production. By allowing operators to practice in both controlled, virtual environments and real-world conditions, the system may enhance the learning experience.

2.2.2 Assembly Task Guidance

Task guidance is another critical application of AR and MR technologies in industrial settings, particularly for assisting operators during complex processes such as assembly tasks. By overlaying digital information onto physical objects or environments, AR/MR systems provide step-by-step guidance, ensuring accuracy and consistency in task execution.

Makris et al. [15] present an example of AR application in task guidance by developing an AR module that delivers step-by-step assembly instructions. Their approach features an assembly sequence generation algorithm, which derives detailed assembly sequences and steps from CAD data and supplementary engineering inputs.

This algorithm provides the necessary information—such as the correct sequence of actions, insertion paths, and components—to guide operators through the assembly process accurately. It distinguishes between fasteners and parts, and represents each tool-action pair (e.g., wrench-screwing) as a reusable template within the AR virtual instruction repository. These templates are dynamically retrieved by the system whenever the corresponding instruction is required during assembly.

By leveraging these templates, the system reduces the cognitive load on both engineers and operators, streamlining the assembly process and enhancing overall efficiency.

Another study by Makris et al. [16] explores the design and implementation of an AR tool aimed at enhancing operator performance and safety in hybrid human-robot collaborative industrial environments. The system is designed to provide operators with immersive assembly instructions and real-time production data while increasing their sense of safety when working near large industrial robots. By integrating with a service-based station controller, the AR tool orchestrates the flow of information to the operator based on task execution status. The system was tested in an automotive sector case study, showcasing its potential to improve operator integration with assembly processes.

The implementation of the AR tool involved several key components. A service-based station controller facilitated communication between the system's resources. Operators interacted with the system through a combination of AR glasses and an Android application running on a smartwatch, which allowed them to indicate task completion and toggle production data messages. The system supported three distinct scenarios: human-only tasks, robot-only tasks, and collaborative human-robot tasks. This multi-modal approach demonstrated the flexibility of the system in adapting to various operational needs, while the wearable devices ensured seamless human-robot interaction.

Similarly to Makris et al., Michalos et al. [17] present a system that combines AR glasses and a smartwatch to support seamless information exchange between operators and a robotic system in an automotive assembly setting. The scenario focuses on two main tasks: loading the rear axle and assembling the wheel groups on either side of the axle.

In this setup, operators use AR glasses to visualize real-time information about the robot and task execution directly in their field of view. Meanwhile, the smartwatch serves as a complementary interaction tool, allowing users to, for example, enable or disable manual robot guidance.

Figure 2.4 illustrates how AR and smartwatch interfaces were applied in practice. The AR glasses display task-related information spatially aligned with the real environment (A), while the smartwatch is used to issue control commands to the robotic system (B).

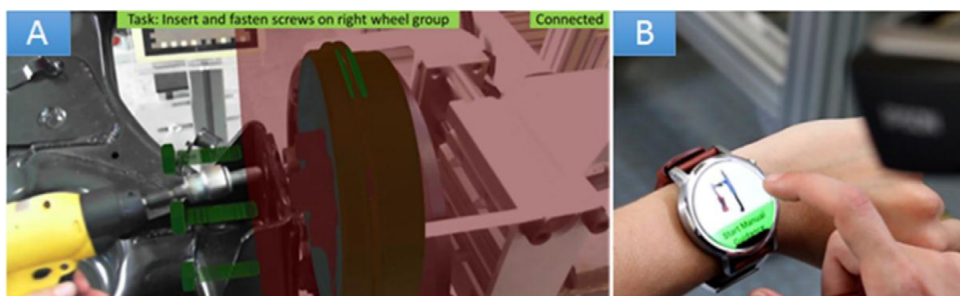


Figure 2.4: AR visualization of task information in real space (A), and smartwatch interface for robot guidance control (B) — taken from Michalos et al. [17].

The AR application developed in this study appears relatively simple, with limited interaction between the virtual and physical elements—a factor that may impact its effectiveness for more involved task guidance scenarios.

Alternatively, other authors have used different methods of interacting with the user. Costa et al. [18] present a novel application of spatial Augmented Reality for guiding operators through small parts assembly tasks. Unlike traditional AR approaches that rely on headsets or mobile devices, this system uses video projectors to overlay textual and video instructions directly onto physical objects in the workspace. This method incorporates a natural interaction interface that allows users to navigate between teaching steps and control video playback using hand gestures detected in the workspace.

In this work, user interaction is achieved through hand gesture recognition, utilizing 3D point cloud data to allow operators to navigate between instructional steps without physical contact with devices. Upon completion of the assembly, the system automatically inspects the final result by estimating the pose of the assembled object and comparing it to a reference 3D model of the correct assembly, leveraging computer vision algorithms for accurate assessment.

A paper by Blaga et al. [19] explores various scenarios where AR can enhance human-robot collaboration, particularly in assembly guidance.

- **Path Visualization for an Automated Guided Vehicle (AGV) — AR Device:** Operators can visualize the planned path of an AGV overlaid onto the real environment. This allows them to detect potential collisions with obstacles and intervene if necessary, improving safety.
- **Robot State Monitoring — AR/MR Headset:** Through AR interfaces, users can monitor a robot's internal states, running processes, and communication data. This enables better situational awareness and system control.
- **Robot Status Tracking and Identification — AR/MR Headset:** The system includes a generic web interface that connects to various types of robots, enabling real-time access to their operational parameters and online variables. Identification is handled through a pretrained database of visual markers, allowing the AR headset to recognize each robot and overlay relevant status onto the real world.
- **Digital twin Visualization — AR/MR Headset:** Real-world assembly components are virtually replicated using CAD models to create digital twins. These digital representations are used to guide operators through the assembly process. By overlaying the virtual parts onto the physical workspace, users can clearly see where each component fits, along with precise positional and orientation cues. This enhances accuracy, reduces errors, and helps operators better understand complex assembly tasks.
- **Small-scale assembly — AR Mobile Device:** AR is used to overlay 3D models onto physical components during small assembly tasks, guiding workers through the process with clear visual aids and step-by-step instructions.

The article is a good example of how AR can support diverse use cases, while also illustrating the range of available technologies and their distinctions. Similar to the work by Michalos et

al., the AR applications presented are relatively simple, relying primarily on informational overlays—aligning more closely with the concept of AR than MR.

Park et al. [20] propose a deep learning-based mobile AR system for intelligent task assistance, which leverages color and depth data for 3D spatial mapping. Unlike traditional marker-based AR methods, their approach eliminates the need for physical markers. It uses deep learning—a sub-field of machine learning that employs artificial neural networks with multiple layers—for instance segmentation, and applies iterative closest point (ICP) algorithms to achieve precise alignment between virtual models and physical objects. The system provides users with augmented 3D instructions projected directly onto real-world objects, enhancing task guidance in manufacturing environments. This innovation marks a significant advancement in AR-based task assistance, particularly in achieving accurate object detection and registration without pre-registration or reliance on markers.

The study emphasizes the advantages of using region-of-interest (RoI)-based spatial mapping, which reduces computational complexity while maintaining high precision. By segmenting specific areas of interest, the system effectively removes noise from point cloud data, enabling seamless integration of virtual and physical objects. Additionally, the authors validate their approach through comparative experiments, showing superior performance in accuracy, task efficiency, and usability over traditional marker-based AR systems. However, the system’s reliance on external hardware for deep learning computations highlights a limitation, as the current mobile devices lack the processing power for on-device execution.

Another study exploring the use of Mixed Reality is presented by Fathi et al. [21], who developed an MR application to support a series of tasks involved in the assembly of a component within a waste management pump. At this workstation, operators are required to assemble small parts such as screws, grommets, and washers onto a larger cast iron component (see Figure 2.5).



(a) Example of a workstation setup in the MR environment.



(b) MR instructions guiding the operator through the assembly process.

Figure 2.5: Mixed Reality interface for assembly support — adapted from Fathi et al. [21].

The workflow for defining a workstation begins with analyzing its process flow, understanding the overall assembly procedure, and identifying the key tasks—particularly if they have not been previously documented. These tasks are then broken down into subtasks that reflect the specific movements performed by the operator.

A notable feature of the system is the "workstation editor," which allows users to create, modify, and browse workstation definitions. Within each workstation configuration, users can adjust tasks, subtasks, and trigger objects—the visible or invisible markers in the MR environment that determine the progression of the task flow. The editor facilitates rapid creation of task hierarchies without requiring a physical visit to the workstation; however, site visits may still be necessary for precise positioning and calibration of the trigger objects.

Experimental results revealed several limitations related to hardware performance, particularly with hand tracking and object detection. These challenges were especially pronounced when working with small components or under suboptimal lighting conditions, highlighting areas for improvement in MR-based assembly support systems.

Though the topic of using AR/MR for task assistance seems to be becoming more relevant in present say, it is not a completely new concept. A paper from 2008 [22] discusses an AR-based approach to assembly guidance, featuring a virtual interactive panel and a visual hierarchical tree structure as central components. The virtual interactive panel features virtual buttons that provide detailed assembly instructions, which operators can activate using a specialized interaction pen tracked in real time by a neural network. The hierarchical tree structure organizes assembly information and sequences in an accessible visual format, simplifying the retrieval and management of instructions during the assembly process. Together, these elements form an AR-based system that eliminates the need for physical markers or sensors on assembly components, offering a flexible and intuitive solution for manufacturing environments.

2.2.3 Human-Robot Collaboration

Another concept relevant to operator training and guidance is human-robot collaboration (HRC). Aligning with the principles of Industry 5.0 (discussed in Chapter 1), many modern workstations are hybrid environments where humans and robots work side by side. HRC seeks to enhance the efficiency and safety of industrial tasks by integrating human adaptability with robotic precision, thereby optimizing workflows, reducing errors, and creating safer shared workspaces. In this context, Mixed Reality technologies, often combined with Digital Twins, are increasingly employed to simulate, train, and assist operators effectively in these complex environments.

In this context, Choi et al. [23] integrate AI and DT technologies with MR to develop an innovative safety-aware HRC framework. This framework combines deep learning with a digital twin to enable real-time safety monitoring and task optimization. The system dynamically calculates safety distances between humans and robots, thereby enhancing safety and efficiency in shared work environments.

The system, illustrated in Figure 2.6, provides operators with safety information and task assistance through the use of MR glasses. Two color and depth sensors are used to reconstruct the working environment, track the operator's skeletal information, and synchronize the physical robot with its digital twin.

A notable strength of the proposed system is its ability to balance real-time performance and accuracy in safety distance calculations. Unlike other methods that rely heavily on 3D point cloud

data, the approach leverages simplified 3D models and offset-based distance calculations, reducing computational overhead while maintaining high precision. The integration of deep learning-based instance segmentation further enhances the system’s capability to identify and track objects and operators in the workspace. However, the study acknowledges limitations, such as potential inaccuracies in skeletal tracking due to occlusions, and the need for more robust MR hardware to improve the user experience. Despite these challenges, the approach demonstrates significant advancements in HRC safety and usability, with results validated through comparative evaluations against traditional methods.

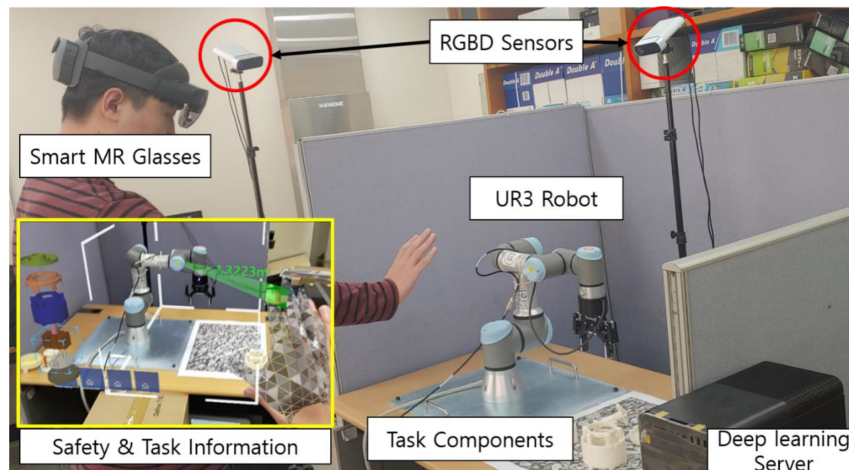


Figure 2.6: Working environment of the approach proposed by Choi et al. [23] for safety-aware human-robot interaction.

Another approach that uses MR to improve human-robot collaboration is presented by Calzone et al. [24]. This application’s main features consist of communicating the robot’s status and intent via a visual feedback via the MR environment.

The system shows a hologram of the robot, anchored on its real counterpart. This calibration procedure is done by using a visual marker in the form of a QR code fixed to the robot. This hologram is used to provide the user with a visual guide of the robot’s joints as well as a projection of the robot’s end effector trajectory.

The article highlights MR’s potential to significantly enhance collaboration and communication between humans and robots in shared workspaces. By providing intuitive visual feedback and real-time information about the robot’s status and intended actions, the application enables operators—regardless of their prior robotics experience—to interact more effectively and safely with robotic systems. This increased transparency and accessibility help to build trust, improve situational awareness, and reduce the risk of accidents, ultimately fostering a safer and more efficient collaborative environment.

In other studies that utilize QR code anchoring, the lack of flexibility poses a significant challenge, as the system should ideally need to accommodate various object types. However, in this application, which is tailored to a specific robot, the requirement for flexibility is less critical.

Building on this work, Carriero et al. [25] introduced bi-directional interaction capabilities between the human operator and the robot. This system is an evolution of the previous one, with the addition of new features that allow the user to not only visualize the robot state and trajectory but also to manipulate it. For example, the user can grab and move the end-effector hologram in the MR environment to change the robot's end-effector path. The system can also check the feasibility of the new trajectory and warn the user if it exceeds the robot's workspace limits.

This application improves upon the previous one by introducing a more interactive experience, transforming the MR environment from a passive information display into an active control interface. This, again, underlines the potential of MR to enhance human-robot collaboration by not only showing critical information to users but also enabling them to interact with and influence robotic systems in real time.

Also building on the optimization of human-robot collaboration, Kousi et al. [26] propose a dynamic framework for designing and reconfiguring HRC assembly lines. Their model, which integrates AI logic, enables the derivation of alternative production system configurations based on real-time data and operational requirements. The researchers leverage digital twins to create accurate virtual representations of the physical assembly environment, optimizing assembly lines by generating optimal configurations in terms of layout and task plans.

The case study demonstrates how incorporating AI and Digital Twins into HRC assembly lines can provide safer and more effective training methods.

Expanding on the integration of MR and DT technologies for HRC, Lalik and Flaga [27] propose a system architecture to enable two-way interaction between a physical robot and its Digital Twin using MR goggles while also measuring the distance between real and virtual objects in real-time.

The system uses MR goggles with depth and stereo cameras to map 3D environments accurately. This enables gesture-based interactions with both physical robots and their digital twins. Operators can control the physical robot by adjusting the DT or make changes to the DT through physical interactions, ensuring seamless bidirectional control.

Supervised by a Programmable Logic Controller (PLC), the system provides precise synchronization and real-time feedback. This approach supports adjustments in industrial workflows and demonstrates the potential of integrating MR and DT technologies.

Additionally, by being able to measure the distance between a virtual and real component, it enables the definition and verification of tasks during the design stage, ensuring safer and more efficient operations involving the robot or its Digital Twin.

The study acknowledges challenges, such as measurement errors due to misalignment between virtual and physical elements and the hardware limitations of current MR devices. Despite these issues, the research highlights the importance of MR in connecting the real world and a digital twin.

Janecký et al. [28] present the design and implementation of a mixed reality human-machine interface for robotic system control. Their interface enables real-time monitoring and control of a robot model through a headset, contributing to a more intuitive and efficient user experience.

The study highlights several benefits of MR integration, including hands-free operation, improved situational awareness, and a more natural interaction paradigm through direct information overlay in the user's field of view. These advantages closely align with the goals of this dissertation, particularly in leveraging MR to enhance operator engagement and usability in industrial environments. However, the authors also note ergonomic limitations—specifically, the current weight of the headset—which may affect user comfort during prolonged use.

2.2.3.1 Fault Detection and Diagnosis

A prominent application of MR and AI technologies is fault detection in manufacturing, where these technologies can significantly enhance accuracy and efficiency.

Wang et al. [29] propose a method in this field that combines deep learning and MR to address challenges in assembly inspection. The system integrates a convolutional neural network with an MR headset to enhance accuracy and efficiency in assembly inspection tasks. By leveraging deep learning for feature extraction and pattern recognition, the system enables real-time identification and localization of assembly faults.

Operators can use gestures or voice commands to interact with the system, which dynamically collects images, processes them via the neural network model, and returns augmented feedback directly into the operator's field of view. This approach eliminates the reliance on manual inspections, which are prone to human error, and introduces a streamlined, data-driven method for improving manufacturing processes.

Experimental results demonstrate high detection accuracy and near real-time response times. However, the study highlights challenges such as the need for extensive training data to prevent overfitting and potential limitations in scenarios with inconsistent lighting or complex object geometries. Nevertheless, this study shows how the integration of deep learning and MR significantly enhances the reliability and usability of fault detection systems in industrial settings.

In a similar vein, an article by Bottani et al. [30] discusses the use of MR in diagnosing and resolving faults within a factory production line. The proposed solution integrates an Android mobile app and a MR application to alert operators to faults and to guide them through the resolution process.

When a machine alert is detected, the devices walk the worker through the necessary steps needed to return the machine to normal function. In fact, the system is able to provide the user with troubleshooting procedures and safety prescriptions as well as the precise location in the plant where the alarm was observed.

The system was tested in an industrial aseptic bottling line consisting of five machines, demonstrating its effectiveness in real working conditions. Notably, it operates without the use of tags, relying instead on the predefined layout of the plant, which enhances its flexibility and adaptability to diverse industrial environments.

2.3 Visualization Mediums

Visualization in MR/AR systems is crucial for presenting information interactively. Common visualization methods include the use of Head-Mounted Displays (HMDs), projectors, and mobile devices. Each medium has its own advantages and limitations, influencing factors such as immersion, usability, and adaptability to different industrial settings.

2.3.1 HMDs

Head-mounted displays are currently the most prevalent visualization medium because they offer operators an immersive, hands-free experience. By overlaying real-time digital content onto the physical world, they allow users to simultaneously interact with their environment and perform other tasks.

2.3.1.1 HoloLens

One of the most prominent examples of HMDs is the Microsoft HoloLens (shown in Figure 2.7), both first and second-generation versions. Across the reviewed literature, this device was employed in thirteen studies [11]–[14], [19], [21], [23]–[25], [27]–[30]. The HoloLens offers several advantages, including its foundation on the familiar Windows 10 operating system, seamless compatibility with Microsoft Dynamics 365 Guides, and close integration with the Mixed Reality Toolkit (MRTK), which was largely developed and optimized alongside the HoloLens 2.



Figure 2.7: Microsoft HoloLens 2 headset

However, it is noteworthy that Microsoft officially discontinued the HoloLens product line in October 2024 [31].

2.3.1.2 Other HMDs

While the Microsoft HoloLens has been the most widely adopted HMD in the literature, its official discontinuation in 2024 has prompted renewed interest in identifying alternative devices.

In the past, devices such as the Epson Moverio BT-200 and BT-300 have been explored in studies focused on human-robot collaboration and task assistance [17], [19].

More recently, the Magic Leap 2 [32], released in 2022 [33], has also emerged as a promising contender.

Nonetheless, no alternative has yet matched the HoloLens 2 in terms of ecosystem integration, developer tools, and widespread industrial adoption. As a result, the search for a robust, flexible, and scalable successor continues.

2.3.2 Mobile Devices

Mobile devices, such as smartphones and tablets, represent another common visualization approach in AR applications [19], [20], [30]. These devices offer portability and ease of use, making them well-suited for task-specific scenarios. However, their primary drawback is the lack of hands-free interaction, which can limit usability in environments where operators need to use both hands.

Additionally, some studies have explored the use of smartwatches [16], [17], which provide quick and lightweight access to contextual information. While convenient, their small screen size restricts their effectiveness for displaying more complex visual content.

2.3.3 Projectors

Spatial AR, which uses projectors to overlay digital content directly onto physical objects, is less common but offers unique advantages. This approach eliminates the need for wearable devices, providing a shared visualization space that multiple users can interact with simultaneously. For example, as mentioned previously, Costa et al. [18] demonstrated the use of a video projector to display textual and video instructions on physical assembly components, presenting a unique solution to the problem.

2.3.4 Summary of Visualization Mediums

Table 2.1 summarizes the visualization technologies used in the reviewed works.

Table 2.1: Summary of Visualization Technologies used in reviewed papers

Study	Visualization Technology Used
Yuan et al. (2008) [22]	HMD
Makris et al. (2013) [15]	Not Mentioned
Makris et al. (2016) [16]	HMD, Mobile Device (Smartwatch)
Wang et al. (2018) [29]	HMD (HoloLens)
Michalos et al. (2018) [17]	HMD (Moverio BT-200), Mobile Device (Smartwatch)
Costa et al. (2019) [18]	Video Projector
Park et al. (2020) [20]	Mobile device (Lenovo Phab 2 Pro)
Blaga et al. (2021) [19]	HMD (HoloLens, Moverio BT-300), Mobile Devices (Tango AR, Android)
Lalik & Flaga (2021) [27]	HMD (HoloLens)
Kousi et al. (2021) [26]	Not Applicable
Bottani et al. (2021) [30]	HMD (HoloLens), Mobile Devices (Android Smartphone and Tablet)
Choi et al. (2022) [23]	HMD (HoloLens 2)
Apostolopoulos et al. (2022) [12]	HMD (HoloLens 2)
Calzone et al. (2023) [24]	HMD (HoloLens 2)
Carriero et al. (2023) [25]	HMD (HoloLens 2)
Verdu et al. (2024) [14]	HMD (HoloLens 2)
Barros (2023) [13]	HMD (HoloLens 2)
Sarhan et al. (2024) [11]	HMD (HoloLens 2)
Janecký et al. (2024) [28]	HMD (HoloLens 2)
Fathi et al. (2024) [21]	HMD (HoloLens)

2.4 Software

The implementation of AR/MR applications relies on a combination of software platforms and engines that enable visualization, interaction, and simulation. During this state-of-the-art review, several key software tools were identified, each offering unique capabilities for creating AR/MR environments and integrating them into industrial applications. This section provides an overview of these tools, focusing on their use in the reviewed literature.

2.4.1 Engines/Simulators

Engines and simulators are essential tools in the development of AR/MR environments and robotics systems. Engines, such as Unity, provide the framework for creating and rendering 3D graphics, managing interactions, and building immersive experiences. Simulators, like Gazebo, focus on replicating real-world physics, sensor data, and environmental conditions, enabling developers to test and validate systems in controlled virtual settings. Both can play a role in designing, evaluating, and deploying AR/MR and robotics solutions.

- **Unity:** Unity is an industry-leading engine that provides tools to create and operate games and other real-time interactive experiences [34]. It is one of the most widely used engines for AR/MR development, known for its versatility and ease of use. Its compatibility with platforms like Vuforia and MRTK makes it a popular choice for applications involving HMDs, such as HoloLens. Many of the reviewed papers [12], [16], [17], [19], [21], [23]–[25], [28], [30] utilize Unity due to its robust support for interactive environments and its ability to integrate seamlessly with hardware like HoloLens.

Unity's large developer community and extensive documentation also contribute to its widespread adoption, offering an accessible entry point for developing complex AR/MR systems. The ability to integrate various libraries and tools, such as physics engines and AI frameworks, further enhances its functionality for industrial applications.

- **OGRE3D (Object-Oriented Graphics Rendering Engine):** OGRE3D is an open-source rendering engine focused on creating high-quality 3D graphics [35]. Unlike Unity, which offers a fully integrated environment for AR/MR development, OGRE3D specializes in rendering tasks, providing developers with fine-grained control over graphics pipelines, shaders, and resource management.

This engine can be effectively combined with Gazebo Simulator to model robotic systems [18]. However, for applications requiring direct AR/MR interactions, other tools like Unity may be more suitable.

- **Gazebo Simulator:** Gazebo is a powerful tool for simulating robotic environments [36]. Frequently used in robotics research, Gazebo provides accurate physics simulation and sensor emulation, making it ideal for testing HRC systems. For example, one of the reviewed

papers [26] utilizes Gazebo Simulator to validate alternative configurations of the assembly system.

Gazebo excels in simulating real-world conditions, allowing developers to test robot interactions in controlled environments. However, it lacks the graphical rendering and user-friendly interface of Unity, making it less suitable for creating immersive AR/MR experiences but invaluable for testing underlying system functionalities.

2.4.2 Platforms

Platforms like Vuforia, Mixed Reality Toolkit, and ARCore play a crucial role in enabling AR applications by providing libraries and tools for object recognition, tracking, and rendering.

- **Vuforia:** A widely used AR development platform that allows for the creation of augmented reality applications. It utilizes computer vision technology to recognize and track 2D images and 3D objects in real time. Its robust tracking capabilities and compatibility with Unity make it a popular choice for creating interactive AR applications [15], [17], [19].
- **Microsoft Mixed Reality Toolkit (MRTK):** MRTK for Unity, developed by Microsoft, is a framework that simplifies creating MR applications on Unity for different platforms like the HoloLens or other MR devices [37]. This toolkit, mentioned in six reviewed studies [11], [12], [21], [24], [25], [28], is an invaluable tool that offers building blocks for spatial interactions, user interface components, and input handling. It also enables rapid prototyping via in-editor simulation that allows you to see changes immediately.

Of particular relevance is the latest generation of the toolkit, MRTK3 [38], which represents a significant advancement towards standardization and cross-platform compatibility. Designed from the ground up to be hardware-agnostic, MRTK3 simplifies the development of applications that can seamlessly transition between devices.

- **ARCore:** Developed by Google, ARCore [39] is an augmented reality SDK that provides cross-platform APIs to build new immersive experiences on Android, iOS, Unity, and Web . It supports motion tracking, environmental understanding, and light estimation. One of the papers [19] utilized ARCore to assist in small-scale assembly with an Android device.
- **ARToolKit:** One of the earliest AR platforms, ARToolKit offers a reliable framework for marker-based AR applications. It was used in Blaga et al. [19] to overlay virtual objects onto the physical environment. Despite being somewhat dated compared to Vuforia and ARCore, it remains a foundational tool in the AR development space.

2.4.3 Ready-to-use MR Applications

In addition to engines and platforms that enable the development of custom AR applications, there are also ready-to-use solutions available. The most common example is **Dynamics 365**

Guides [40], a Microsoft application designed to facilitate the creation and delivery of step-by-step holographic instructions. This tool allows organizations to implement AR-based training and task guidance without the need for extensive programming or development expertise.

Some of the studies reviewed make use of Dynamics 365 Guides as a straightforward solution for delivering assembly instructions [11], [13]. However, this approach is less flexible, as it is tightly coupled with the Microsoft HoloLens platform—which has since been discontinued—thereby limiting its long-term scalability and hardware independence.

2.4.4 Summary of Software

Table 2.2 summarizes the software used in the reviewed papers, highlighting their implementation in AR/MR applications.

Table 2.2: Summary of Software used in reviewed papers

Studies	Software Used
Yuan et al. (2008) [22]	Not Mentioned
Makris et al. (2013) [15]	Not Mentioned
Makris et al. (2016) [16]	Unity with Vuforia
Wang et al. (2018) [29]	Not Mentioned
Michalos et al. (2018) [17]	Unity with Vuforia
Costa et al. (2019) [18]	Gazebo with Ogre3D
Park et al. (2020) [20]	Unity
Blaga et al. (2021) [19]	Unity, Vuforia, ARToolKit, ARCore
Lalik & Flaga (2021) [27]	Not Mentioned
Kousi et al. (2021) [26]	Gazebo
Bottani et al. (2021) [30]	Unity
Choi et al. (2022) [23]	Unity
Apostolopoulos et al. (2022) [12]	Unity with MRTK and OpenCV
Calzone et al. (2023) [24]	Unity with MRTK
Carriero et al. (2023) [25]	Unity with MRTK
Verdu et al. (2024) [14]	Not Mentioned
Barros (2023) [13]	Dynamics 365 Guides
Sarhan et al. (2024) [11]	Dynamics 365 Guides and Unity with MRTK
Janecký et al. (2024) [28]	Unity with MRTK
Fathi et al. (2024) [21]	Unity with MRTK

The table highlights the diversity of software tools employed in AR/MR research. Vuforia stands out as a widely adopted platform for tracking and interaction, while the Unity engine dominates the development landscape due to its flexibility and compatibility with hardware like HoloLens. Additionally, the increasing appearance of MRTK indicates a growing trend in using standardized frameworks for mixed reality development.

Chapter 3

Practical Implementation

3.1 Use Case

To demonstrate the concept of using MR to support operator training and task execution, a prototype application was developed to guide the user through the process of installing and securing a wing mirror onto a van door. This use case was selected due to its relevance as a common industrial assembly task involving both spatial alignment and manual operations, making it well-suited for evaluating the effectiveness of MR-based instruction and feedback.

3.1.1 Material

The van door was chosen due to its availability within the SYSTEC laboratory and its suitability for demonstrating an industrial assembly task in a Mixed Reality training context.

The door model (figured in 3.1), used across several light commercial vehicles (part of the K9 family) manufactured by Stellantis and Toyota, can be found in multiple commercial and passenger vehicle variants, including:

- Peugeot Partner (and Rifter)
- Citroën Berlingo (and Berlingo Multispace/XL)
- Opel/Vauxhall Combo (and Combo Life)
- Toyota ProAce City (and ProAce City Verso)
- Fiat Doblò (third generation, from 2022 onwards)

Its widespread presence in modern vehicles makes it highly representative of real-world use cases.



Figure 3.1: Van door (without the quarter glass piece installed)

Along with the van door, the corresponding wing mirror (seen in Figure 3.2) was available.



Figure 3.2: Wing mirror

Additionally, in order to install this mirror, the door's quarter-glass must be installed, as that is where the mirror is fastened to. Figure 3.3 shows the van door with this piece installed.



Figure 3.3: Van door with quarter glass piece installed

3.1.2 Assembly Steps

As mentioned before, the assembly to be aided is the placement of the wing mirror onto the door. This assembly effectively has two steps:

- Place the wing mirror correctly, ensuring the cable is properly aligned.
- Fasten the screws located on the back side of the door to secure the mirror.

When placing the mirror onto the attachment part, the operator needs to pay special attention to the wing mirror's cable. This needs to be passed through the hole in the attachment part of the quarter-glass window.



Figure 3.4: Correct cable position, passed through the attachment piece on the quarter glass.

After passing the cable through, the mirror can be attached. In Figure 3.5, we can see the mirror correctly attached to the door.



Figure 3.5: Mirror correctly placed onto door

Once the mirror is attached to the door, it should be secured to the back of the door using screws on the holes marked in Figure 3.6.

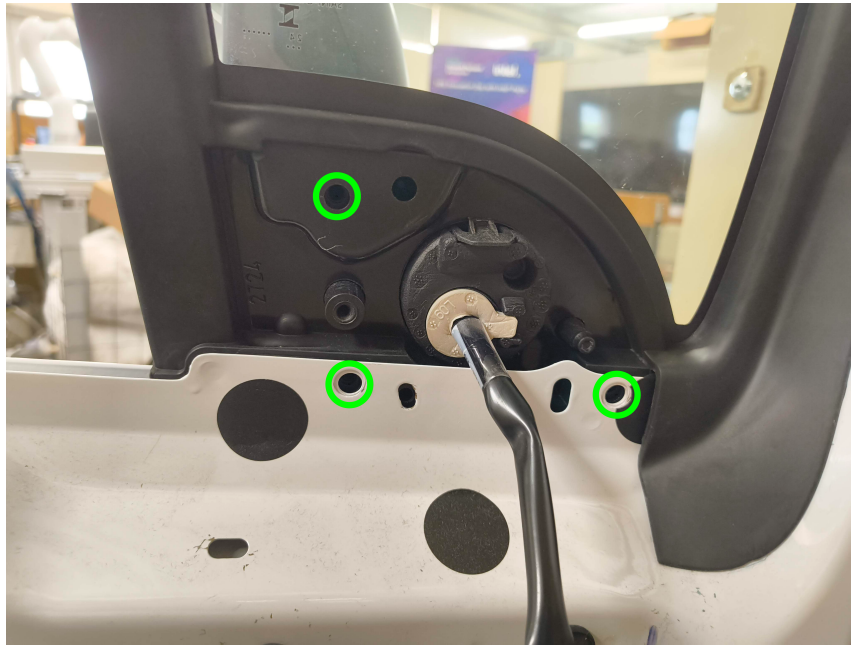


Figure 3.6: Screw hole location (seen from back of door)

The assembly is then complete.

3.2 Development

3.2.1 Tools

For this work, the HoloLens 2 served as the primary hardware platform. Despite its discontinuation, it remains an appropriate device for MR development. Moreover, with the advancements introduced in MRTK3 and its growing shift toward hardware-agnostic development, applications will be able to be deployed across a variety of MR devices, not limited to the HoloLens.

In terms of software, Microsoft Dynamics 365 Guides is a viable option for creating step-by-step, business-oriented MR applications. However, it is designed exclusively for the Microsoft HoloLens platform [40]. Given the headset's discontinuation and the increasing demand for device-independent solutions, more flexible and scalable development approaches are preferred.

In this context, Unity was selected as the primary development engine for the custom MR application. Unity offers extensive documentation, a large developer community, and strong compatibility with critical MR development tools such as MRTK and Vuforia.

By integrating Unity with Vuforia, the application can track 3D objects, such as the van door, from multiple angles. This object-based tracking removes the need for static markers like QR codes, which are commonly used in other studies to anchor virtual content. Markerless tracking is particularly beneficial in industrial environments where attaching QR codes to each physical component is impractical or undesirable. Vuforia was chosen over alternatives like OpenCV due

to its seamless integration with Unity and its focus on AR use cases, making it simpler and more practical for rapid development of MR applications.

MRTK3 also played a vital role in simplifying MR application development. It provides a comprehensive suite of tools, reusable components, and interaction models—including spatial input, UI (User Interface) elements, and interaction logic. These features significantly reduce development complexity, enabling faster prototyping and easier deployment across diverse MR platforms.

3.2.1.1 Testing and Simulation

A major advantage of using Unity in combination with MRTK and Vuforia is the ability to test and iterate on application logic directly within the game engine. Unity's built-in *Play Mode* allows developers to simulate runtime behavior without deploying to a physical device. Notably, Vuforia supports the simulation of real-world objects, enabling the detection of Model Targets virtually. This allows for testing, tracking logic, and interactions without requiring physical components or repeated deployments to the HoloLens.

Additionally, MRTK provides built-in hand interaction simulation, making it possible to test gesture-based controls and UI elements within the Unity editor. This greatly accelerates development by enabling rapid prototyping and debugging in a controlled environment.

An alternative testing approach explored was the use of the HoloLens Emulator[41], which simulates a HoloLens device on a PC. However, this option was ultimately discarded. Play Mode offered all the necessary capabilities for development—including simulated tracking—and the emulator introduced unnecessary overhead and performance issues.

3.2.1.2 Software Tool Versions

To ensure reproducibility and maintain consistency across development and testing, it is important to specify the exact software environment used. Table 3.1 lists the main software tools, frameworks, and SDKs employed throughout the implementation process, along with their corresponding versions.

It should be noted that, according to Microsoft, "MRTK3" refers to the third generation of the Mixed Reality Toolkit rather than a specific version number.

Table 3.1: Software tools and versions used in development

Tool	Version/Generation
Unity	6.0.0 (6000.0.38f1)
Visual Studio	2022 (Version 17.14.13)
Mixed Reality Toolkit	MRTK3
Vuforia Engine	11.3.4
Windows SDK	10.0.19041.0

3.2.2 3D Models

To enable detection and tracking of key components for the assembly task, as well as to support training with virtual elements, 3D models of both the van door and the wing mirror were produced.

For simplicity and ease of use, the scanning of the objects was done with an iPhone. Starting with the iPhone 12 Pro, Apple devices are equipped with LiDAR sensors [42], which enable the capture of reasonably accurate 3D scans of real-world objects. The app *AR Code Object Capture* [43] was used to perform the scans.

While the resulting models are suitable for prototyping, in an industrial context, access to CAD files of the actual objects would probably allow for higher precision and model fidelity.

Post-processing was performed in Blender to correct minor issues in the scanned models. For instance, the wire on the mirror model—irrelevant for tracking purposes—was removed. Additionally, Blender was used to export the final models in the *.fbx* format, which is compatible with Unity and suitable for integration into the Mixed Reality environment.

Figure 3.7 and figure 3.8 show the obtained 3D models for the van door and the mirror, respectively.



Figure 3.7: Door 3D model



Figure 3.8: Wing mirror 3D model

A 3D model of the quarter-glass window attachment piece was also produced. While this component is not tracked in the application, it can be displayed as a hologram to help illustrate the

real assembly scenario or to support training when the physical part is unavailable. The resulting model is shown in Figure 3.9.



Figure 3.9: Quarter-glass window attachment piece 3D model

3.2.3 Object Tracking

As previously mentioned, the Unity game engine was used for the development of the MR application, along with Vuforia and MRTK.

3.2.3.1 Door and Mirror Tracking

Vuforia is responsible for detecting and tracking the 3D model of the van door and the wing mirror.

To enable this, the scanned 3D models were converted into Vuforia Model Targets using the Model Target Generator [44] tool provided by Vuforia. Model Targets are a digital representation of a physical object — created from a 3D CAD or mesh model — that Vuforia uses to detect and track the real-world version of that object in the camera view [45].

For this implementation, an Advanced Model Target Database was used. This type of database allows for the detection of objects from all sides. It enables recognition from 360 degrees around the object without requiring the user to align an outline with the physical item to initiate tracking [46].

Despite these advantages, a key limitation remains: Vuforia can only track one Model Target at a time. As a result, simultaneous detection of multiple components is not supported, restricting the handling of assemblies involving several real objects. In this use case, it meant that either the door or the mirror could be tracked at a given moment, but not both concurrently. Workarounds to mitigate this issue are discussed later in the document.

3.2.3.2 Attachment Piece Tracking

In addition to Model Targets, the use of an Image Target was considered for detecting the attachment piece of the quarter-glass window. As illustrated in Figure 3.4, this component is significantly smaller and flatter than the entire door. It was hypothesized that detecting a smaller,

more localized area might improve tracking precision and stability. Another advantage is that Image Targets [47], which are 2D representations of printed or digital images, can be tracked simultaneously, alongside other targets.

However, this tracking method is optimized for recognizing flat, image-based content, such as posters, labels, or documents. Given this, the Image Target approach was ultimately deemed unsuitable for this specific use case.

3.2.4 Training Modes

In order to accommodate different levels of available materials or operator proficiency, three training modes were created:

- Virtual Training
- Mixed Training
- Full Training

These modes differ in which elements are real and tracked by Vuforia or simply virtual holograms that the user can interact with – performing the same actions as if it were physical. Table 3.2 shows a summary of which elements are tracked or virtual depending on the mode.

Table 3.2: Components status' in different training modes

Mode	Door		Mirror	
	Tracked	Virtual	Tracked	Virtual
Virtual		✓		✓
Mixed	✓			✓
Full	✓		✓	

Virtual Training simply places the virtual elements into the real world. The user interacts with fully virtual counterparts of the van door and wing mirror. This allows the user to practice the assembly without any risk to actual equipment.

Mixed Training mode serves as a middle ground, utilizing a real door, tracking it, and having the user perform the assembly with a virtual mirror.

Full Training utilizes both the real door and the mirror. Simply aiding the operator with a real assembly.

3.2.4.1 Scene and Mode Management

These modes are selected in a Start Menu scene, which is the first scene loaded when the application starts. The user can choose their desired training mode from this menu before proceeding to the main training environment.

To make this transition between the Start Menu and the Main Scene, a script was created `ModeManager.cs` to handle the scene transitions. The script utilizes Unity's built-in scene management system, which allows for the loading and unloading of different scenes within the application.

This script contains a public variable, `TrainingMode`, which is an enumeration defining the three training modes: Virtual, Mixed, and Full. When the user selects a mode from the Start Menu, the corresponding method in `ModeManager.cs` is called, setting the `TrainingMode` variable accordingly and then loading the Main Scene.

This public variable is also accessible from other scripts in the Main Scene, allowing them to adjust their behavior based on the selected training mode.

3.2.4.2 Start Menu

The Start Menu scene provides a simple UI element for mode selection (seen in 3.10). This menu component was created using a pre-made UI element available with MRTK3.

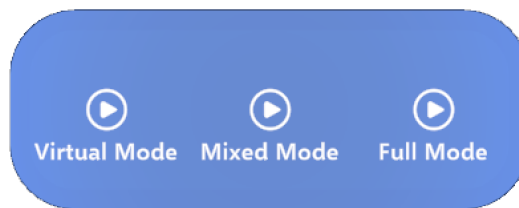


Figure 3.10: UI element used in the Start Menu to select the training mode.

The user can press one of the three buttons to select their desired training mode. Each button is linked to a specific method in the `ModeManager.cs` (Listing A.1) script, which then sets the training mode and transitions to the Main Scene.

3.2.5 Main Scene

The Main Scene is where the core functionality of the MR application takes place. It encompasses the 3D models, tracking components, user interface elements, and interaction logic necessary to guide the user through the assembly process.

3.2.5.1 Scene GameObjects

In Unity, each scene is composed of *GameObjects*, which represent the fundamental building blocks of the scene and can be anything from 3D models and lights to UI elements or logic controllers.

The Main Scene in this project is structured around several key *GameObjects*, each serving a distinct role in the application's functionality.

Assembly GameObjects

Figure 3.11 illustrates the hierarchy of the Main Scene, specifically the GameObjects used for tracking and assembly logic.

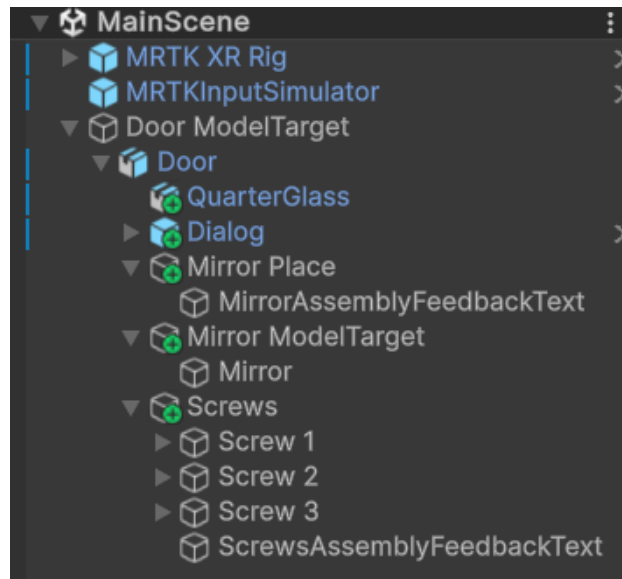


Figure 3.11: Hierarchy of the Main Scene, highlighting the primary GameObjects used in the MR application.

Among these GameObjects are the door, mirror, and a variety of UI elements that support the training experience. Particularly noteworthy are the Model Target GameObjects, which represent the physical objects that Vuforia must track. These objects contain the necessary Vuforia components for detection and tracking, ensuring that the virtual content is accurately aligned with the physical environment.

The door object is implemented as a child of its corresponding Model Target, with the same hierarchy applied to the mirror. Each of these objects encapsulates both the assembly logic and the 3D model of the component, which is displayed whenever the physical object is not present. This hierarchical structure ensures that, when an object is successfully tracked, its logic object automatically inherits the position and orientation of the Model Target. Conversely, when tracking is disabled or not required, the corresponding Model Target can be deactivated and the logic/3D object detached, supporting a modular architecture that clearly separates assembly logic from tracking functionality.

Additionally, the scene includes several *Feedback* objects, which provide textual cues to guide users through the assembly process. Both the mirror and the screws have their own UI elements as children, ensuring that the feedback follows the position and orientation of the relevant object during the interaction. There is also a *Dialog* object attached to the Door, which is used to prompt the user for confirmation at specific points in the assembly process.

MRTK GameObjects

The Mixed Reality Toolkit (MRTK) used in this work provides several essential GameObject *prefabs*—preconfigured templates containing a defined set of components—that enable mixed

reality interactions and user interface functionality within the application.

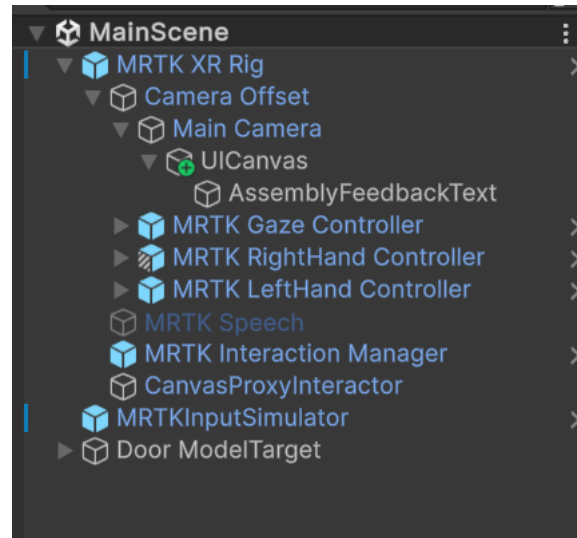


Figure 3.12: MRTK GameObjects in the Main Scene, responsible for mixed reality interactions and UI elements.

Figure 3.12 illustrates some of these key GameObjects used in the Main Scene, namely:

- *MRTK XR Rig*: The central GameObject that serves as the core hub for all functionality related to the user’s MR headset and input. Its transform represents the user’s head, and it is responsible for managing the camera, determining the user’s viewpoint, and handling input sources such as gaze, gestures, and voice commands.
- *MRTK Input Simulator*: Enables hand interaction simulation within the Unity editor, facilitating testing without a physical device.

The Main Camera is parented to the MRTK XR Rig, which ensures that its position and rotation match the user’s head movement. A child of this camera is the *UICanvas* GameObject, which serves as a plane for placing UI elements such as the *AssemblyFeedbackText*. By parenting the canvas to the camera, we ensure critical feedback remains consistently visible in the user’s field of view.

3.2.5.2 Scene Flow and States

To guide the user through the assembly process, the application employs a state machine architecture. This design allows for clear definition and management of different stages in the assembly workflow, ensuring that the user receives appropriate instructions and feedback at each step.

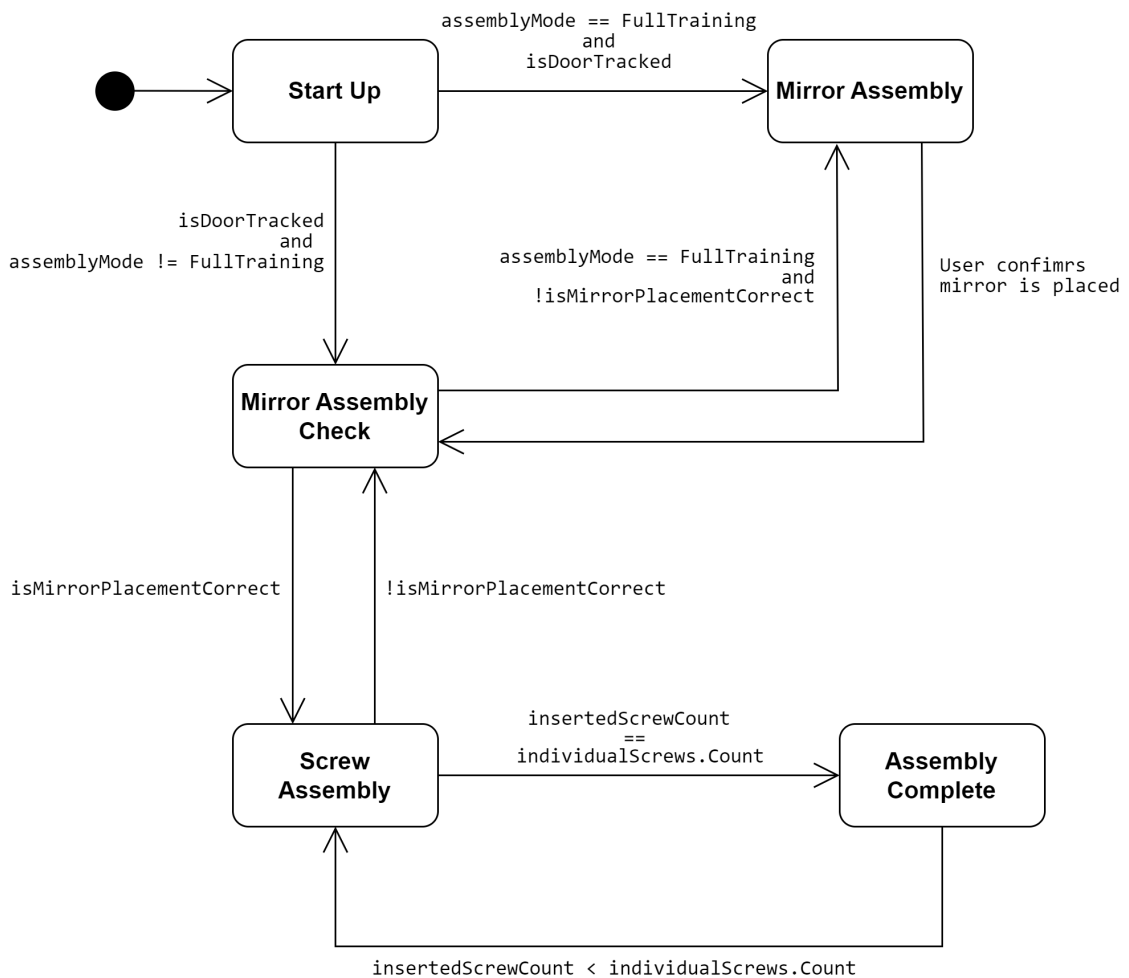


Figure 3.13: UML State Machine Diagram showcasing the revised behavior and states of the main scene of the MR application.

This state-based structure is illustrated in Figure 3.13 and consists of the following states:

- **Start Up:** The application begins by detecting and tracking the van door. In the case of Virtual Training mode, the door is virtual and thus considered tracked by default. In Mixed Mode, Vuforia actively searches for the physical door, and, once found, the system transitions to Mirror Assembly Check state - the same happens in Virtual Training. For Full Training, once the door is tracked, we move onto the Mirror Assembly state.
- **Mirror Assembly:** This state is only visited in Full Training mode. The objective is for this state to be when the system waits for the user to confirm they have placed the mirror. The system prompts the user with an animation showing how to place the mirror correctly and a button to confirm when they have done so. Once the user confirms this action, the system transitions to the Mirror Assembly Check state.
- **Mirror Assembly Check:** In this state, regardless of training mode, the system verifies whether the mirror's position matches the expected location. If the mirror is correctly

placed, the process advances to the next stage of the assembly; otherwise, it either waits for it to be correct (in Virtual and Mixed modes) or returns to the Mirror Assembly state (in Full Training mode).

- **Screw Assembly:** With the mirror correctly placed, the user proceeds to attach the three screws. These are represented by interactive holograms, that the user can insert. Once all screws are placed, the system transitions to the final state.
- **Assembly Complete:** Once both screws are placed correctly, the final state is reached. A completion message is displayed, indicating the successful conclusion of the training session.

3.2.5.3 Object Tracking and Detection

Originally, the idea was to sequentially locate the door using Vuforia, then the mirror, and subsequently guide the user through the assembly steps. During this process, the system would continuously track both objects and provide real-time feedback on the mirror position.

However, this approach revealed two key limitations: Vuforia doesn't allow for the simultaneous tracking of two Model Targets, and it demonstrated inconsistent performance when continuously tracking the moving mirror Model Target.

To address this, the final implementation tracks the door until the system transitions to the mirror placement step. At that point, the user confirms that the mirror has been placed, and the system performs a single verification of the mirror's position rather than continuously tracking it. This strategy avoids the need for simultaneous multi-object tracking while still ensuring placement accuracy.

In modes other than Full Training, this restriction does not apply. During Virtual and Mixed Training, the mirror exists only as a hologram, allowing the system to provide continuous feedback on its position without relying on Vuforia.

Tracking status of objects is managed via the variables `isDoorTracked` and `isMirrorTracked`, which are either set to true by default (for virtual objects) or updated dynamically through callback methods when the status of the Model Targets changes. This decentralized, event-driven approach avoids repeated polling or central status checks, resulting in better performance and a more modular system design.

3.2.5.4 Assembly Management

The door object — the center of the assembly — contains a script named `AssemblyManager.cs` (Listing A.2), which orchestrates the overall assembly process. This script manages the state transitions, user prompts, and feedback throughout the training experience.

3.2.5.5 Mirror Placement

The mirror placement process is managed by a custom script, `MirrorAssembly.cs` (Listing A.3), attached to the mirror object. This script is responsible for providing visual guidance, verifying correct positioning, and communicating completion to the central assembly logic.

To guide placement, it provides a visual guide utilizing a "ghost" hologram of the component—shown in Figure 3.14—serving as a target position. This ghost model is a semi-transparent counterpart of the 3D model of the wing mirror described in Subsection 3.2.2.

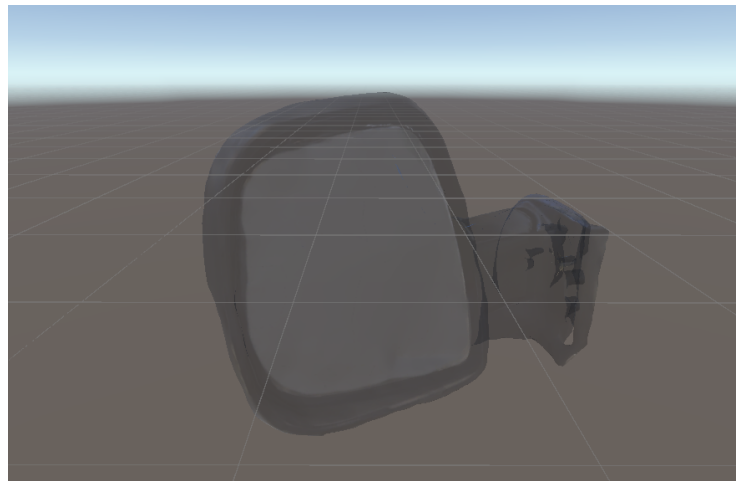


Figure 3.14: "Ghost" counterpart of the wing mirror model used as a placement guide.

In Full Training mode, the ghost hologram is displayed with an animation illustrating the correct placement procedure. This animation is dismissible via a dialog prompt (shown in Figure 3.15), allowing the user to watch it as many times as needed before proceeding, while not interfering with the view of the real door and mirror.

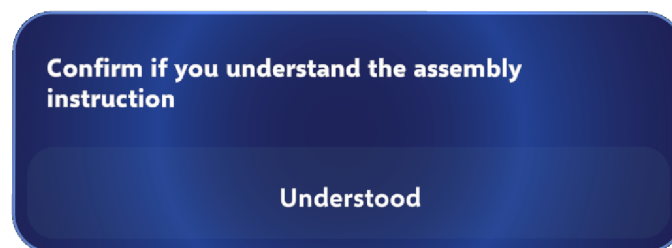


Figure 3.15: Dialog prompt asking the user to confirm they have understood the mirror placement instructions.

Once the user confirms they have understood the instructions, the ghost hologram is hidden and the dialog prompt changes, waiting for the user to confirm that the mirror has been placed (Figure 3.16). This step is required in Full Training mode, as the system cannot automatically verify the mirror's position without continuous tracking.

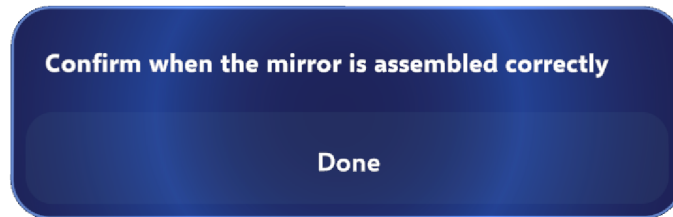


Figure 3.16: Dialog prompt asking the user to confirm they have placed the mirror.

Once the user confirms the mirror has been positioned, the mirror object notifies the `AssemblyManager` to transition to the next state, where the system verifies the mirror's placement.

This verification is performed using the mirror's `CheckMirrorPlacement()` method, which compares the mirror's current position and orientation against the target defined by the ghost hologram. If the placement is within acceptable thresholds, it is considered correct. It also knows the previous status, so it only notifies the `AssemblyManager` when the status changes, preventing repeated notifications.

In Virtual or Mixed Training mode, the ghost hologram is shown continuously, showing the correct position and orientation for the mirror. The `CheckMirrorPlacement()` method is run non-stop, allowing the user to receive real-time feedback - via a `MirrorAssemblyFeedbackText` element attached to it - as they manipulate the mirror object, allowing them to adjust its position until it aligns with the ghost.

Regardless of training mode, once proper alignment is achieved, it automatically notifies the `AssemblyManager` that the placement has been completed. This is done via a method call to `UpdateMirrorAssemblyStatus()` in the `AssemblyManager` script, which then handles the state transition to the next phase of the assembly process.

In all instances the place object is hidden once the mirror is correctly placed, ensuring a clear view of the assembly.

3.2.5.6 Screw Placement

Once the mirror is correctly positioned, the user must secure it using three screws, as outlined in Subsection 3.1.2. Each screw hole includes a visual guide to indicate the correct placement

location, like the one shown in Figure 3.17.

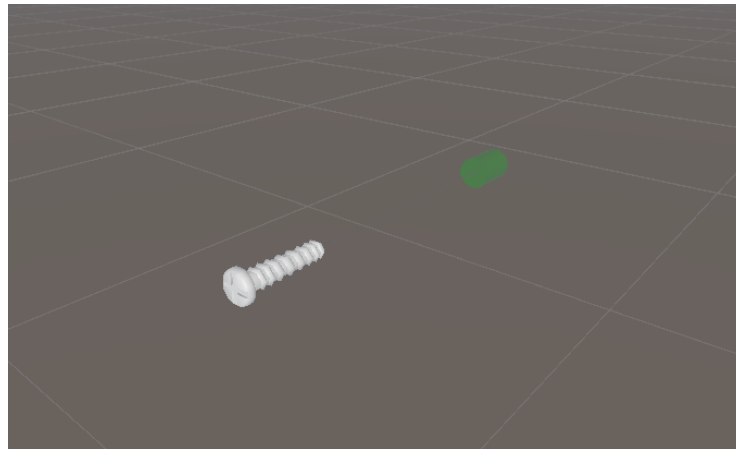


Figure 3.17: Screw hologram and visual guide object for each screw hole.

The guide consists of three elements:

- Screw Hologram
- Hole Highlight
- Screw-Hole Line

The hole highlight and connecting line visually associate each screw with its designated hole. These elements were created using simple 3D cylinders with semi-transparent materials to ensure they are noticeable yet unobtrusive.

The screw hologram itself is interactive, allowing the user to "click" it and insert it into the highlighted location. The screw was made using an asset from the Unity Store [48].

Each screw guide includes a `StatefulInteractable` component from MRTK3, which enables gesture-based interaction. This component detects user input, such as clicks or hand gestures, and triggers corresponding events when the object is selected.

Interaction and feedback are handled by the `ScrewInteractive.cs` (Listing A.4) script attached to each screw guide hologram. When a user selects a screw, the `StatefulInteractable` component notifies this script, which then calls `UpdateScrewAssemblyStatus()` on the `AssemblyManager`. The assembly manager updates the screw count and checks whether all three have been secured. In parallel, the script provides real-time visual feedback by moving the screw hologram into the "inserted" position and updating the feedback text to reflect the current number of screws secured.

After all three screws are secured, the `AssemblyManager` transitions to the final state and displays a completion message, indicating the successful end of the assembly task.

Real-Time Screw Insertion Feedback

In addition to providing a visual guide for screw insertion, the possibility of implementing real-time feedback on the insertion of physical screws was also considered. However, this functionality

was not tested within the scope of this dissertation. Given the small size and limited distinctive features of screws, it is expected that Vuforia would face significant challenges in reliably detecting and tracking them.

3.2.5.7 Text Feedback and Dialog Prompts

Text Feedback

To guide the operator and convey system status, the application includes several on-screen text elements that provide real-time feedback throughout the training experience. Two helper classes were developed to support this functionality:

- *Billboard* – ensures that 3D text elements in the environment always face the user, improving legibility.
- *TextDisplay* – provides utility methods to display messages either temporarily or persistently, as well as to define the color of the text.

All text elements in the scene make use of these scripts to ensure they are always oriented towards the user and can be easily updated with new messages as needed.

The *Billboard* class, implemented in *Billboard.cs* (Listing A.5), rotates the text object to face the camera every frame. This is achieved using Unity's `LookAt()` method, which aligns the forward vector of the text object with the direction to the camera. To maintain proper orientation, the script also zeroes out any rotation around the x and z axes, keeping the text upright.

The *TextDisplay* class, implemented in *TextDisplay.cs* (Listing A.6), provides two main methods for displaying text: `ShowTemporaryText()` and `SetPersistentText()`. The `ShowTemporaryText()` method displays a message for a specified duration before clearing it, which is useful for transient notifications or short status updates. Conversely, the `SetPersistentText()` method sets a message that remains visible until explicitly changed, making it ideal for ongoing instructions or persistent feedback. Both methods also allow specifying the text color, enabling visual cues to indicate different message types (e.g., warnings in red or confirmations in green).

Dialog Prompts

As discussed in Subsection 3.2.5.5, dialog prompts are used when we require user input to proceed. Namely, when the system requires the user to confirm that they have understood the mirror placement instructions (Figure 3.15), and when it requires confirmation that the mirror has been positioned (Figure 3.16).

These prompts are implemented using the *Dialog* prefab provided by MRTK3, which is a pre-built UI component designed for displaying dialog boxes. This prefab contains a title, message text, and buttons for user interaction. It also includes built-in functionality for handling button clicks and dismissing the dialog.

To adapt this functionality to the needs of the application, a script with a new subclass named *StaticDialog* was developed. Unlike the base *Dialog* class from the prefab, which follows

the user's gaze and is dismissed immediately upon button selection, the `StaticDialog` remains fixed in place and persists across multiple prompts. This modification was necessary because the default gaze-following behavior was unintuitive, and automatic dismissal conflicted with the need to reuse the same dialog for more than one confirmation. This subclass is implemented in the `StaticDialog.cs` script (Listing A.7).

3.2.6 Application Deployment

In order to test the application on the HoloLens 2 device, the project must be built and deployed to the device. This process involves two main steps:

- *Building on Unity*: The project must first be built on Unity, which generates a Visual Studio solution containing all the necessary files for deployment.

On Unity 6, this is done on the build profiles window (shown in Figure 3.18), where the build settings must be selected, paying attention to these specific points:

- *Scene List*: The correct scenes that one wishes to build must be selected, paying attention to the index presented on the right. Number 0 corresponds to the scene that is loaded upon startup.
- *Architecture and SDK Version*: Something that must also be paid attention to is the architecture to deploy to. The HoloLens 2 runs on ARM64 architecture, so it is essential to select "ARM64" as the target architecture in the build settings. The Windows SDK version should match the one installed on your system and supported by the HoloLens 2 (e.g., 10.0.19041.0).
- *Build Configuration*: It is important to set this as Release always, as the MRTK documentation states there are known problems with the Debug option with MRTK.

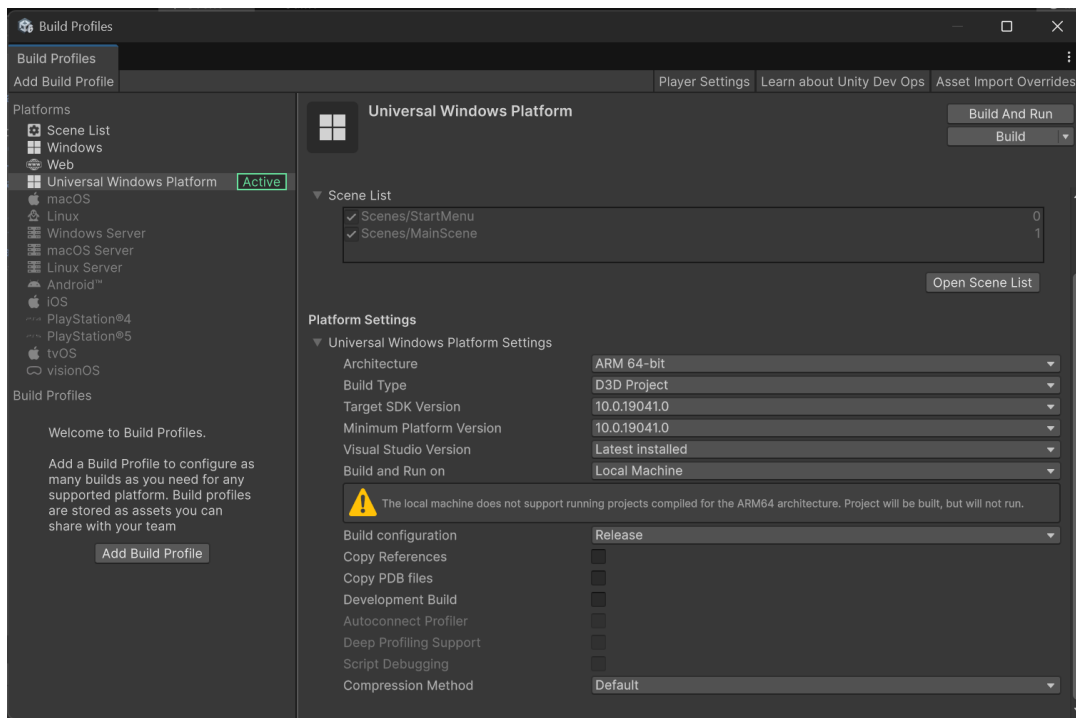


Figure 3.18: Unity build profiles window

- *Deployment on Visual Studio:* After building the project in Unity, the generated Visual Studio solution (.sln) can be opened. From there, the application can be deployed to the HoloLens 2 device.
 - *Release build:* To deploy the application on Visual Studio as a Release build, i.e. no script debugging, one must select "Release" on the top bar (shown in Figure 3.19), and then select "Start Without Debugging" (or press Ctrl + F5) to deploy the application to the device. This method installs the app on the HoloLens 2 and launches it without attaching the debugger, which is generally more stable for MRTK applications.
 - *Debug build:* If debugging is desired, one can select the "Debug" option on the top bar (shown in Figure 3.19), and then select "Start Debugging" (or press F5) to deploy the application to the device and attach the debugger. This allows for breakpoints, variable inspection, and step-through debugging directly on the HoloLens 2.

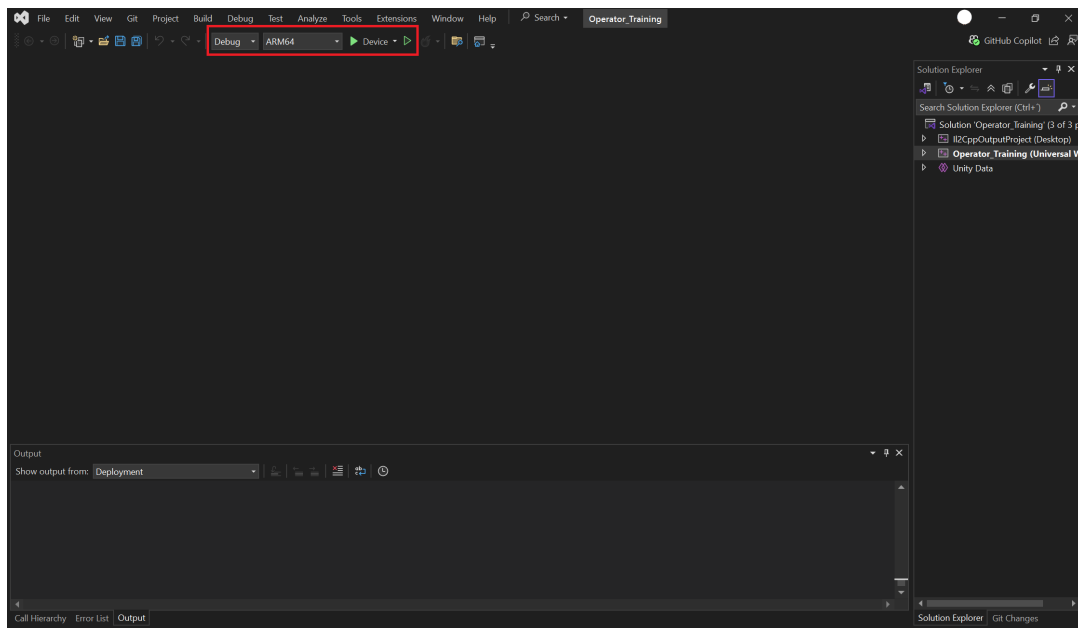


Figure 3.19: Deployment process in Visual Studio — with the top bar highlighted in red

3.2.6.1 Deployment Issues

Throughout the development cycle, several common issues in deploying the application were encountered. Mismatched SDK versions between the Unity configuration and the Visual Studio environment frequently caused build failures. A second common issue was project corruption due to stale data in the Unity build cache, which could be resolved by clearing the cache and performing a clean rebuild of the project.

3.2.7 Version Control and Project Repository

For the development of this project, Unity’s Version Control was utilized to manage changes and maintain a history of the project. This tool integrates with Unity, allowing for seamless versioning of assets, scenes, and scripts.

However, it doesn’t allow for public repositories, so the project files cannot be shared directly. Instead, the source code for this application is made available on GitHub¹. This repository contains the Unity project files, including all scripts and assets used in the implementation.

¹https://github.com/toastermctoast/Operator_Training

Chapter 4

Experimental Results and Discussion

4.1 Simulation in Unity

4.1.1 Unity's Play Mode

In addition to the final deployment on the HoloLens headset, the training application can also be executed and tested directly within the Unity engine using Play Mode. In addition to aiding in development and testing, this feature also allows end users to run and interact with the application as if it were live, without the need for physical deployment. This feature offers an alternative way for users to practice the assembly tasks, particularly when access to MR headsets is limited.

The application could also be adapted into a standalone desktop version, enabling training on conventional computers. This would broaden accessibility and reduce hardware requirements, making the solution more scalable in industrial environments where MR devices are less available than PCs.

This simulation environment also serves as a Digital Model of the real assembly workstation. While it offers less *hands-on* experience compared to using physical components, it accurately replicates the task sequence and spatial logic of the real-world procedure, making it a valuable tool for both training and system testing.

4.1.1.1 Controls

When using the application in Unity's Play Mode, users can interact with the system using standard mouse and keyboard inputs. The mouse cursor serves as a pointer to select UI elements and interact with holograms, while keyboard keys facilitate navigation and control within the application.

The following controls are implemented:

- **Right Mouse Button Hold + Mouse Movement:** Rotate the camera view around the scene.
- **Space Bar Hold + Left Mouse Button Click:** Hold the space bar and click to select UI buttons and interact with holograms.

- **W, A, S, D Keys:** Move the camera forward, left, backward, and right, respectively, allowing users to navigate around the scene.
- **Q and E Keys:** Move the camera down and up, respectively, providing vertical navigation.

4.1.1.2 Model Target Simulation

As discussed in Subsection 3.2.1.1, Vuforia's Model Target emulation feature is utilized to simulate the detection and tracking of physical objects within Unity's Play Mode. This allows for testing the application's logic and interactions without requiring actual physical components.

In modes where object tracking is necessary, the objects appear in white and require the user to visually focus on them to trigger detection. This behavior is illustrated in Figures 4.1b (Mixed Mode) and 4.1c (Full Mode).

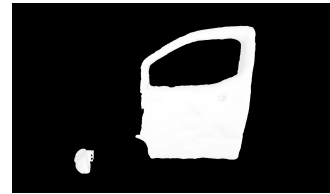
In contrast, for Virtual Training, where object tracking is not required, the 3D components appear directly in place, as illustrated in Figure 4.1a.



(a) Assembly objects in Unity's Play Mode - Virtual Mode.



(b) Assembly objects in Unity's Play Mode - Mixed Mode



(c) Assembly objects in Unity's Play Mode - Full Mode

Figure 4.1: Mirror Assembly Phase in Unity's Play Mode with different training modes: (a) Virtual Mode, (b) Mixed Mode, and (c) Full Mode.

To manage the visualization of Model Targets in Play Mode, the user can enable or disable the "Model Target Visualization" option in the Model Target Preview script within Unity's inspector, as shown in Figure 4.2.

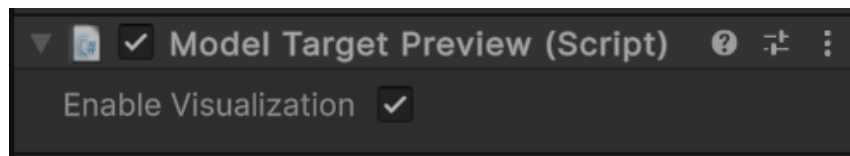


Figure 4.2: Enabling Model Target Visualization in Vuforia's configuration settings within Unity.

When enabled, Vuforia displays the Model Target in white, simulating its detection. When disabled, the Model Target is hidden, allowing users to see the virtual holograms as they would appear in the actual MR experience.

4.1.2 Start Menu

As mentioned in 3.2.4, the system begins by prompting the user to select a training mode. This start menu behaves the same in Unity's Play mode as well as with the HoloLens. Figure 4.3

shows a snapshot of the Start Menu scene, where the user interacts with the Start Menu UI mode selection element. The user can hover over each button and click to select the desired mode.

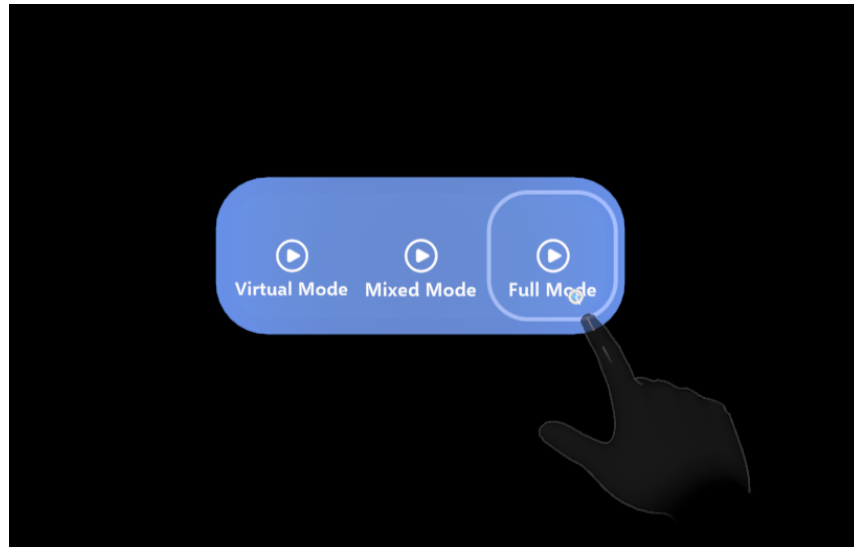


Figure 4.3: Start Menu scene within Unity's Play Mode

4.1.3 Main Scene

Unity's Play Mode then loads and proceeds to the Main Scene.

If the current training mode requires door tracking, this requirement is met by asking the user to look at the door model, as shown in Figure 4.4.

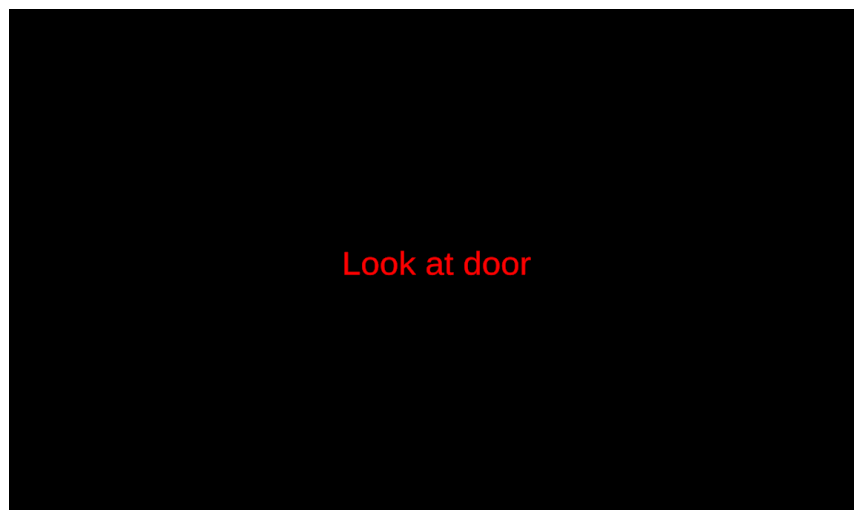


Figure 4.4: Prompting the user to look at the door for detection in Mixed Mode (and Full Mode) within Unity's Play Mode.

In Play Mode, tracking is assumed once the user looks at the simulated object, as mentioned in Subsection 4.1.1.2.

Other than this detail, as outlined previously, the logic and appearance of the Main Scene are effectively identical for virtual and mixed modes. Given this, in the next section, for simplicity, images will be taken from the virtual mode.

4.1.3.1 Assembly Process in Virtual and Mixed Modes

The assembly process begins with the placement of the mirror. The user is shown the mirror ghost place along with real-time feedback through its text element. Figure 4.5 shows how the system informs the user if the mirror isn't yet correctly placed, by giving information on whether the position or orientation is wrong.

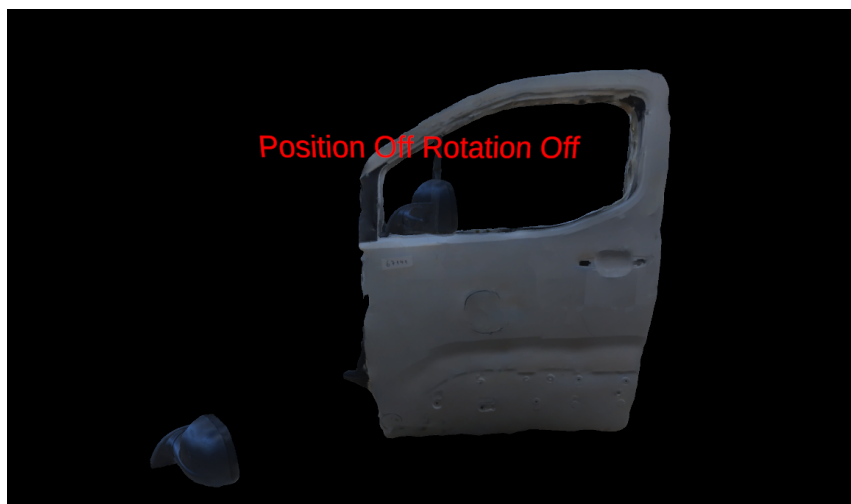


Figure 4.5: Real-time feedback in Unity's Play Mode: the system displays a ghost mirror and on-screen text to guide correct placement.

This allows the user to adjust the placement in an iterative fashion, giving real-time feedback.

Once the mirror is placed, the system displays a "Mirror placed correctly" text, as shown in Figure 4.6.



Figure 4.6: Text feedback in Unity's Play Mode for correct mirror placement

Then, the user is instructed to go to the back of the door to place the virtual screws with an "Insert screws" text, shown in Figure 4.7



Figure 4.7: Text feedback in Unity's Play Mode to insert the screws.

To insert the screws, the user must click on the holograms, which, once detected by the system, moves the hologram to the hole and removes the green hole highlight.

Once one of the screws is inserted, this is confirmed to the user with a "1/3 screws inserted" text, shown in Figure 4.8.



Figure 4.8: Text feedback in Unity's Play Mode when one out of three screws is inserted.

When two screws are inserted, the text updates to "2/3 screws inserted", as shown in Figure 4.9.



Figure 4.9: Text feedback in Unity's Play Mode when two out of three screws are inserted.

Once all three screws are inserted, the user receives an "Assembly Complete" text, seen in Figure 4.10.



Figure 4.10: "Assembly Complete" text shown in Unity's Play Mode once assembly is complete.

Video demonstrations of the assembly process in Unity's Play Mode are available online for both the Virtual¹ and Mixed² modes.

4.1.3.2 Assembly Process in Full Mode

The Full Training mode, as discussed in Chapter 3, introduces specific behaviors compared to the other modes.

Once the door is tracked (via Vuforia simulation in Play Mode), the user is shown an animation of the mirror ghost being inserted into its correct position on the door. The animation illustrates the placement procedure, moving the hologram from an offset position (4.11a) to its final alignment (4.11b).



(a) Initial position of the mirror ghost in Full Mode within Unity's Play Mode.



(b) Final position of the mirror ghost in Full Mode within Unity's Play Mode.

Figure 4.11: Mirror ghost animation in Full Mode within Unity's Play Mode: (a) initial position, and (b) final position on the door.

The animation can be dismissed by the user through a dialog (Figure 4.12).

¹https://youtu.be/nQv_5NX9icQ

²<https://youtu.be/hp89zXMmxHg>



Figure 4.12: Dialog for the user to dismiss the mirror ghost animation in Full Mode within Unity's Play Mode.

Once dismissed, a second dialog appears (Figure 4.13), prompting the user to confirm when the mirror has been physically placed.



Figure 4.13: User confirming they have placed the mirror via the dialog in Full Mode within Unity's Play Mode.

This triggers mirror detection, during which the system instructs the user to look at the mirror (Figure 4.14).



Figure 4.14: Prompting the user to look at the mirror for detection in Full Mode within Unity's Play Mode.

Once tracked, the system verifies the mirror's position and orientation. If incorrect, text feedback instructs the user to "Place mirror correctly" (Figure 4.15), and the process returns to the Mirror Assembly state described in Subsection 3.2.5.2.



Figure 4.15: Text Feedback in Unity's Play Mode for incorrect mirror placement in Full Mode.

The ghost animation and confirmation dialogs are then repeated. When the placement is correct, the user receives confirmation feedback (Figure 4.16).

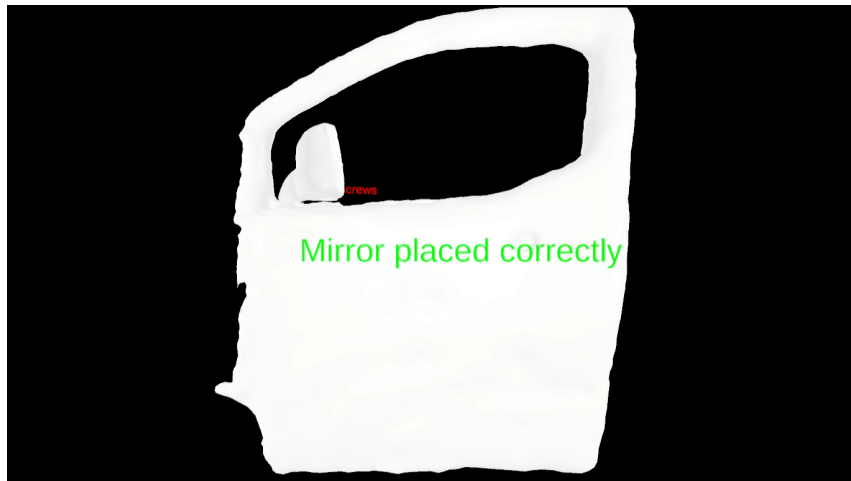


Figure 4.16: Text Feedback in Unity's Play Mode for correct mirror placement in Full Mode.

After successful placement, the process continues with screw insertion, which operates identically to the other modes.

A video demonstration of the Full Mode assembly process in Unity's Play Mode is available online³. In the video, the user first incorrectly places the mirror, receiving feedback to adjust it. After correcting the placement, the user proceeds to insert the screws, completing the assembly. Since this is done in Unity's Play Mode, it is necessary to move the simulated mirror model target to the correct position instead of simply moving the hologram.

4.2 HoloLens Deployment

The system functions in a very similar way when using the HoloLens to complete an assembly.

4.2.1 Start Menu

At the start of the application, the user is also prompted to select a mode and can do so by air clicking on the desired option, as shown in Figure 4.17.

³<https://youtu.be/1QOcxm7RHeI>

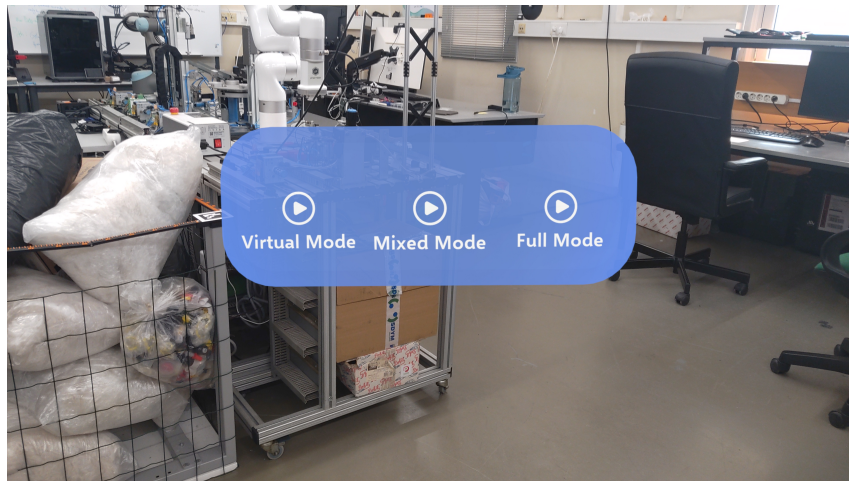


Figure 4.17: Start menu element in HoloLens deployment

4.2.2 Main Scene

After mode selection, the Main Scene loads.

The scene's overall appearance and logic in HoloLens deployment closely mirror the Unity Play Mode configuration. The key difference is the holographic overlay: depending on the selected mode, holograms are either overlaid onto the real environment or real-world objects are tracked to anchor the content.

4.2.2.1 Virtual Mode

For the case of the virtual mode, there is no object tracking or detection. The virtual holograms of the elements are simply placed onto the real world, as shown in Figure 4.18.



Figure 4.18: Virtual elements placed onto the real world in Virtual Mode.

The Virtual Mode experience on HoloLens closely mirrors its behavior in Unity's Play Mode, as detailed in Subsection 4.1.3.1.

The assembly workflow, therefore, remains unchanged, following the same main steps:

- Place the mirror with the support of the ghost hologram guide and its text feedback (Figure 4.19).
- Insert the screws by clicking on their holograms with text feedback:
 - "Insert screws" text instruction (Figure 4.20).
 - After the first screw is inserted, "1/3 screws inserted" text feedback (Figure 4.21a).
 - After the second screw is inserted, "2/3 screws inserted" text feedback (Figure 4.21b).
 - After the third screw is inserted, "All screws inserted" text feedback (Figure 4.21c).
- Assembly completion feedback (Figure 4.22).

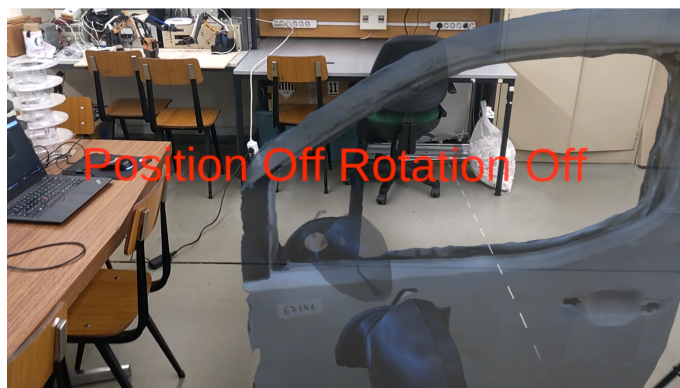


Figure 4.19: Mirror placement guided by ghost hologram and text feedback in Virtual Mode.



Figure 4.20: Prompt in Virtual Mode guiding the user to insert holographic screws in HoloLens deployment.

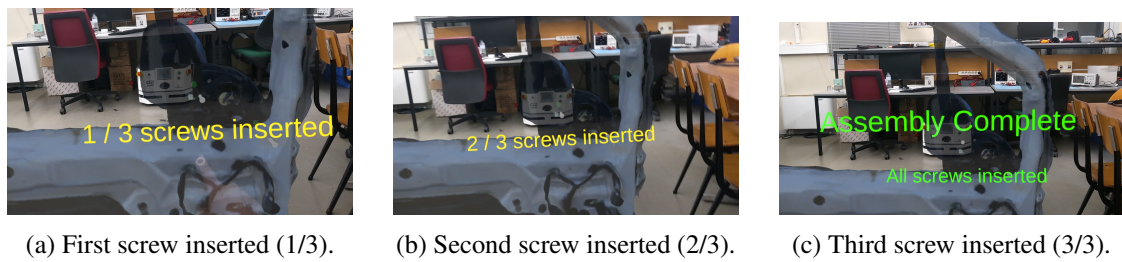


Figure 4.21: Text Feedback in Virtual Mode when screws are inserted in HoloLens deployment: (a) first screw, (b) second screw, and (c) all screws inserted.

In Figure 4.21c, both the "All screws inserted" text and the "Assembly Complete" text are visible, as the former is displayed immediately after the last screw is inserted, accompanied by the latter to indicate the completion of the entire assembly process.



Figure 4.22: "Assembly Complete" text shown in Virtual Mode once assembly is complete in HoloLens deployment.

A video demonstration of the Virtual Mode assembly process using HoloLens 2 is available online⁴.

4.2.2.2 Mixed Mode

For Mixed Mode, the assembly process on HoloLens closely resembles Virtual Mode, but with the addition of object tracking for the door. The steps are as follows:

- The user is prompted to look at the door to enable tracking (Figure 4.23).
- Once the door is tracked, the mirror ghost hologram appears, giving the user real-time text feedback to assist with correct mirror placement (Figure 4.24).
- After the mirror is placed correctly, the user receives confirmation feedback (Figure 4.25).
- The user is prompted to insert the screws (Figure 4.26).

⁴<https://youtu.be/j52Y1g-LF-o>

- As each screw is inserted, the system updates the text feedback to indicate progress (Figure 4.27).
- Upon inserting all screws, the user receives an "Assembly Complete" message (Figure 4.28).

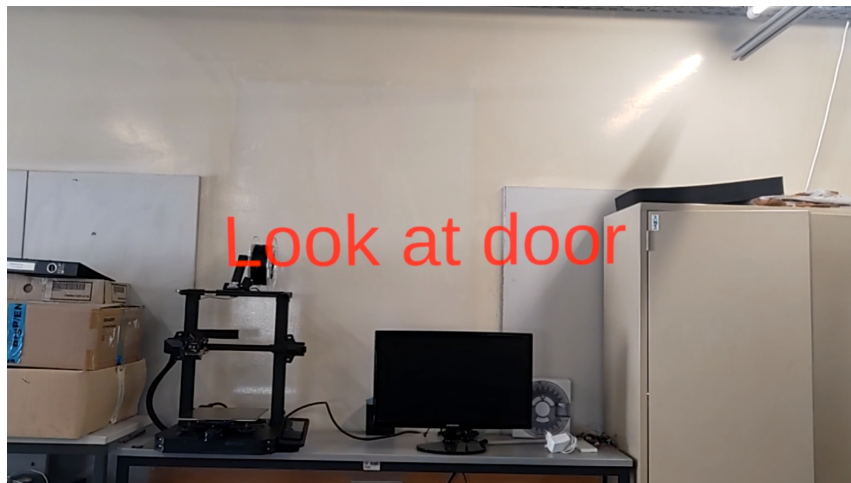


Figure 4.23: Prompting the user to look at the door for detection in Mixed Mode.



Figure 4.24: Mirror placement guided by ghost hologram and text feedback in Mixed Mode.



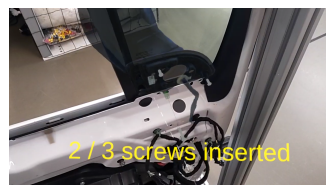
Figure 4.25: Text feedback in Mixed Mode for correct mirror placement in HoloLens deployment.



Figure 4.26: Prompt in Mixed Mode guiding the user to insert holographic screws in HoloLens deployment.



(a) First screw inserted (1/3).



(b) Second screw inserted (2/3).



(c) Third screw inserted (3/3).

Figure 4.27: Text feedback in Mixed Mode when screws are inserted in HoloLens deployment: (a) first screw, (b) second screw, and (c) all screws inserted.



Figure 4.28: "Assembly Complete" text shown in Mixed Mode once assembly is complete in HoloLens deployment.

A video demonstration of the Mixed Mode assembly process using HoloLens 2 is available online⁵.

4.2.2.3 Full Mode

Compared to Virtual and Mixed Modes, Full Mode introduces additional steps due to its specific requirements mentioned previously. The workflow proceeds as follows:

- The user is prompted to look at the door to enable tracking (Figure 4.23 — the same as Mixed Mode).
- An animation shows the mirror ghost moving into position (Figure 4.29).
- The user dismisses the animation via a dialog (Figure 4.30).
- The user confirms when the physical mirror has been placed (Figure 4.31).
- The user is prompted to look at the mirror to enable tracking (Figure 4.32).
- The system checks the mirror's placement and provides feedback if needed (Figure 4.33).
- Once correct, the user proceeds to insert the screws (Figure 4.34 and Figure 4.35).

⁵<https://youtu.be/xnR1mA2XYdI>



(a) Initial position of the mirror ghost animation in Full Mode.



(b) Final position of the mirror ghost animation in Full Mode.

Figure 4.29: Mirror ghost animation in Full Mode within HoloLens deployment: (a) initial position, and (b) final position on the door.



Figure 4.30: Dialog for the user to dismiss the mirror ghost animation in Full Mode within HoloLens deployment.



Figure 4.31: User confirming they have placed the mirror via dialog in Full Mode within HoloLens deployment.



Figure 4.32: Prompting the user to look at the mirror for detection in Full Mode within HoloLens deployment.



(a) Text feedback when the mirror is incorrectly placed in Full Mode.



(b) Text feedback when the mirror is correctly placed in Full Mode.

Figure 4.33: Mirror placement feedback in Full Mode within HoloLens deployment: (a) incorrect placement, and (b) correct placement.



Figure 4.34: Prompt in Full Mode guiding the user to insert holographic screws in HoloLens deployment.



Figure 4.35: Text feedback in Full Mode when screws are inserted in HoloLens deployment: (a) first screw, (b) second screw, and (c) all screws inserted.

The video demonstration of this mode is available online⁶.

4.2.3 Limitations and Challenges

In addition to the successful results presented, some limitations were encountered during the development and testing phases.

Misalignment issues arose when aligning holograms with physical objects, particularly in Mixed and Full Modes, where interactions between real and virtual elements are critical. In these modes, the mirror ghost hologram occasionally appeared slightly offset to what should be the real mirror position, as shown in Figure 4.36.

Similarly, during the screw insertion step, the green highlights around the screw holes were not always perfectly aligned with the physical holes due to tracking inaccuracies. This misalignment—illustrated in Figure 4.37—could reach several centimeters. Furthermore, as the user moved around the assembly, the holograms sometimes shifted or drifted relative to the physical geometry. A video demonstration showing how the holographic screws appear to drift relative to the real holes as the viewpoint changes is available online⁷.

Such misalignments may lead to confusion during the training process, as users might struggle to accurately place the mirror or screws based on the holographic guides.

⁶<https://youtu.be/M-POhXieLWM>

⁷<https://youtu.be/8TjvJ6C1lB8>



Figure 4.36: Example of misalignment: the mirror ghost hologram appears offset from the actual mirror position in Full Mode.



Figure 4.37: Example of misalignment: the holographic screws do not perfectly align with the physical holes in Full Mode.

The root cause of the misalignments is not immediately clear but could stem from a combination of factors:

- Vuforia Model Target tracking limitations, which can be affected by lighting conditions, viewing angles, and the geometric complexity of the tracked objects. In some cases, Vuforia may lose track of the object or miscalculate its position, leading to noticeable offsets in

hologram placement. This is particularly evident when the user moves around the object, causing the holograms to shift relative to the physical geometry.

- Inaccuracies in the 3D models used for tracking, which may not perfectly match the physical objects in terms of scale or detail. For example, as shown in Figure 4.38, the 3D model lacks precision around the screw holes compared to the real door. Such discrepancies could contribute to tracking errors, as Vuforia relies on the 3D model's geometry for recognition and pose estimation.
- Separate modeling of the door, quarter glass and mirror, which introduces minor errors in their relative positioning. Because the mirror and quarter glass placement in relation to the door were manually estimated in Unity, this may have introduced small alignment errors. A composite model of the door with and without the mirror assembled could improve virtual alignment and reduce these discrepancies. This may also explain part of the observed misalignment in both the screw holes and the mirror ghost holograms.

Additionally, the mirror is a particularly challenging object to track due to its highly reflective surface, which can confuse the device's spatial tracking algorithms. Its tracking is usually performed from the object's back face, as reflections often create misleading visual cues that severely interfere with accurate pose estimation.



(a) Real van door with visible screw holes

(b) 3D model of the van door with screw holes

Figure 4.38: Comparison between the real van door (a) and its 3D model (b), highlighting differences around the screw holes.

Another important challenge to mention is the matter of holograms possibly impeding user's view of the real objects, especially in Full Mode. For instance, when guiding the screw insertion step, the holographic screws and highlights could obscure the user's view of the actual screw holes on the door. This could lead to confusion or difficulty in aligning the physical screws with the holes, particularly if the holograms are not perfectly aligned due to tracking inaccuracies.

To mitigate this, the holograms were made semi-transparent, and during mirror placement, the user is prompted to dismiss the ghost animation before proceeding. However, further refinements

could improve both visibility and usability. In particular, for the screw guide holograms, adjusting their opacity, outline, or overall design could help ensure that virtual elements do not obscure important real-world features during assembly.

4.3 Discussion

The implementation of this use case demonstrates the feasibility and potential of employing Mixed Reality to enhance operator training in industrial assembly tasks. The developed application effectively guides users through the sequential process of assembling a wing mirror onto a van door, combining spatially aligned holograms with real-time interaction feedback. This supports the broader goal of fostering immersive, hands-free learning experiences in industrial contexts.

To contextualize the contribution of this work, it was compared with relevant literature on AR/MR-based task training and guidance (Table 4.1). The comparison focuses on the tracking approach employed and the presence of multiple training modes.

Table 4.1: Comparison between studies and the present work.

Study	Object Tracking Method	Multiple Training/Assistance Modes
Yuan et al. (2008) [22]	Deep learning-based (for pen-component interaction)	No
Makris et al. (2016) [16]	Marker-based (Vuforia)	No
Wang et al. (2018) [29]	Deep learning-based	N/A
Michalos et al. (2018) [17]	Marker-based (Vuforia)	Yes
Costa et al. (2019) [18]	Markerless (point cloud alignment via feature matching and ICP)	No
Park et al. (2020) [20]	Deep learning-based point cloud segmentation and ICP	No
Blaga et al. (2021) [19]	Marker-based (ARToolkit, Vuforia)	No
Bottani et al. (2021) [30]	Markerless (predefined spatial anchoring using HoloLens spatial mapping)	N/A
Choi et al. (2022) [23]	Markerless (deep learning and ICP)	N/A
Apostolopoulos et al. (2022) [12]	Markerless (OpenCV-based 2D tracking)	Yes
Calzone et al. (2023) [24]	Marker-based	N/A
Carriero et al. (2023) [25]	Marker-based	N/A
Verdu et al. (2024) [14]	Marker-based	Yes
Barros (2023) [13]	Marker-based	No
Sarhan et al. (2024) [11]	Not Mentioned	Yes
Fathi et al. (2024) [21]	Markerless (predefined spatial anchoring using HoloLens spatial mapping)	No
This work	Markerless (Vuforia)	Yes

A key strength of the developed system lies in its support for multiple training modes—Virtual, Mixed, and Full—allowing the experience to be tailored to different levels of material availability and training progression. Virtual mode enables complete simulation without the need for physical components, making it especially useful for initial practice or remote learning environments. Mixed mode combines physical and virtual elements, allowing users to develop spatial awareness

with real-world objects while maintaining flexibility. Full mode offers the most realistic experience, guiding users through the actual assembly using real components and MR overlays.

Of the works reviewed, only a few [11], [12], [14] support multiple training modes. For example, Verdu et al. [14] present a system that is designed to be used in the design phase, during commissioning and implementation, and along the production phase. Apostolopoulos et al. [12] and Sarhan et al. [11] also introduce systems with multiple modes; however, these modes do not represent varying levels of real and virtual integration for the same assembly task, as achieved in this work.

In contrast, the present system uniquely enables seamless switching between Virtual, Mixed, and Full modes — as well as desktop simulation — within a unified workflow, allowing users to progress from fully simulated to fully physical training scenarios. Play Mode also supports complete execution of the application—making it possible to practice the full training sequence on a PC without an MR headset or physical materials. This capability expands the training’s accessibility and offers a fallback when headsets are limited or unavailable.

Another defining feature of this work is the use of markerless object detection through Vuforia Model Targets. This approach removes the need for physical markers such as QR codes, which are often impractical in industrial environments where attaching such markers to components may not be feasible. Unlike most studies reviewed, the system identifies and tracks the actual geometry of the door and mirror, enabling object recognition from a wide range of viewing angles.

Furthermore, the system does not merely recognize the object in two dimensions, as observed in some of the referenced works; instead, it estimates the object’s full three-dimensional pose (position and orientation) in space. This capability is essential for accurately aligning holograms with physical objects, where precise spatial relationships are critical for effective operator training.

While this approach offers clear benefits, it also revealed some limitations. Vuforia’s current inability to track multiple Model Targets simultaneously required workaround strategies—such as assuming a fixed position for one object while tracking another. Moreover, occasional tracking inaccuracies led to misalignments between holograms and physical objects, as discussed in Subsection 4.2.3.

One thing this work does not address is the evaluation of training effectiveness. While the system provides an immersive and interactive training experience, its actual impact on learning outcomes, skill retention, and transfer to real-world tasks remains to be empirically validated.

From a development standpoint, integrating Unity, MRTK, and Vuforia proved advantageous. Unity’s and Vuforia’s simulation tools significantly accelerated iteration cycles by enabling functional testing without full deployment to the HoloLens 2. However, the hardware itself introduced challenges: prolonged headset use occasionally caused user fatigue due to weight and limited field of view. These ergonomic factors remain a common limitation of current head-mounted displays.

Finally, performance on the HoloLens 2 was generally stable, though minor frame drops occurred during recording. This may stem from the concurrent processing demands of real-time tracking, rendering, and interaction handling. While these did not critically affect usability, further optimization could enhance fluidity in more complex future scenarios.

In summary, this work demonstrates a practical and adaptable MR-based training framework that advances beyond traditional AR assembly guidance systems by combining markerless 3D tracking and multi-mode adaptability within a unified environment. These results reaffirm MR's potential for scalable, flexible, and immersive operator training while highlighting key areas for continued improvement in tracking robustness and hardware ergonomics.

Chapter 5

Conclusion and Future Work

5.1 Conclusions

This dissertation set out to explore the use of Mixed Reality and Digital Twin technologies to improve the training of newly hired operators in industrial assembly environments. By using immersive tools and spatially anchored guidance, the proposed solution aimed to reduce the training gap, minimize risk, and enhance learning outcomes through hands-free and interactive experiences.

The developed prototype—a custom MR application built with Unity, Vuforia, and MRTK—demonstrates the feasibility of combining these technologies to simulate and support real-world assembly operations. Specifically, it enables operators to follow a step-by-step procedure for assembling a wing mirror onto a van door through holographic guidance and real-time feedback.

A notable strength of the system is its adaptability: three distinct training modes—Virtual, Mixed, and Full—allow the same task to be performed across different levels of physical and digital integration. This flexibility accommodates varying material availability and training stages, offering both fully virtual and physically assisted learning environments.

Furthermore, Unity’s Play Mode significantly accelerated development and testing by enabling simulation and debugging without the need for continuous deployment to the HoloLens 2. This also enables the system to function as a fully desktop-based training tool, improving accessibility in contexts with limited MR hardware availability.

The implementation of markerless object tracking using Vuforia Model Targets proved particularly valuable. This approach eliminates the need for markers (like QR codes), providing a more practical and flexible solution for dynamic industrial settings. It also allows object recognition from a wide range of viewing angles by tracking the actual 3D geometry of physical components rather than relying on 2D image matching.

Nevertheless, certain limitations were identified. The markerless tracking occasionally suffered from instability, leading to misalignment between holograms and physical objects, especially during mirror placement and screw insertion in Mixed and Full Modes. These deviations were likely caused by limitations in Vuforia’s tracking accuracy and sensitivity to viewing angles,

as well as minor geometric inaccuracies in the 3D models used for recognition and alignment. Moreover, the current version supports tracking only one object at a time, which limits scalability to multi-component assemblies. Feedback was also restricted to text-based cues, leaving room for richer, multimodal interaction in future iterations.

Despite these challenges, the system demonstrates a significant step forward in leveraging MR and Digital Twin technologies for industrial training. The modular architecture and use of widely adopted development tools (Unity, Vuforia, MRTK) ensure that the solution can be further extended and adapted to a variety of industrial scenarios.

5.2 Future Work

Building upon the foundation established in this dissertation, several directions can be pursued to enhance and extend the current system, both in functionality and applicability:

- **Improved Object Tracking:** Investigating the integration of alternative or hybrid tracking approaches could address several limitations identified in this work, including Vuforia's occasional tracking instability, sensitivity to viewing angles, and inability to track multiple objects simultaneously. Combining Model Targets with deep learning-based object recognition trained on dedicated datasets could improve robustness and precision, and enable the reliable tracking of more complex assemblies with small or low-feature components such as screws. Additionally, tracking smaller subcomponents—such as the quarter glass itself—could enhance spatial accuracy for holograms anchored relative to these elements, improving the overall alignment between virtual and physical objects.
- **Improved 3D Model Fidelity:** To mitigate alignment errors, future work should include a plan to source industrial CAD files or use high-precision 3D scanning equipment to ensure that the virtual models used for tracking are a perfect match for the physical components.
- **User Evaluation:** Conducting user studies with real operators or trainees in industrial contexts would provide valuable feedback on usability, learning effectiveness, and engagement. This would also help validate the system's impact compared to traditional training methods.
- **Enhanced Feedback and Interaction Systems:** The current system relies primarily on visual and text-based feedback. Expanding this to incorporate multimodal feedback—such as voice guidance, auditory cues, or more comprehensive visual progress indicators—could enhance user engagement, comprehension, and situational awareness. Furthermore, introducing natural interaction modalities, including voice commands, could make the training process more intuitive, efficient, and accessible for a wider range of users.
- **Multi-Platform Support:** While the current prototype was developed specifically for the HoloLens 2, its discontinuation necessitates extending compatibility to other Mixed Reality headsets. Supporting devices—such as the Magic Leap or other future MR devices—would

significantly improve the system's portability and long-term viability. Given that MRTK3 already supports a robust, cross-platform development framework, future iterations could leverage this capability to ensure a consistent, device-agnostic user experience.

- **Desktop Training Variant:** Since Unity's Play Mode allows for full application simulation, a dedicated desktop version of the training system could be formalized as a lightweight alternative, useful for early-stage learning or training centers with limited MR hardware.

The directions outlined above represent a clear roadmap for advancing the current MR prototype toward a validated, production-ready training solution. By prioritizing robust, multi-object tracking and committing to rigorous user validation, future efforts can systematically address the technical and usability limitations identified in this study. Ultimately, continued development in these areas will solidify the system's potential to fundamentally improve training efficiency, safety, and operator competence in complex manufacturing environments, positioning this research as a significant contributor to the evolution of industrial training within the Industry 4.0 and 5.0 paradigms.

References

- [1] K. Schwab, *The Fourth Industrial Revolution*. World Economic Forum, 2016, ISBN: 9781944835019. [Online]. Available: www.weforum.org
- [2] “Industry 5.0 towards a sustainable, human-centric and resilient european industry,” Tech. Rep. DOI: [10.2777/308407](https://doi.org/10.2777/308407) [Online]. Available: <https://ec.europa.eu/eurostat/statistics->
- [3] P. Milgram and F. Kishino, “A taxonomy of mixed reality visual displays,” *IEICE Transactions on Information Systems*, pp. 1321–1329, 12 Dec. 1994. [Online]. Available: http://vered.rose.utoronto.ca/people/paul_dir/IEICE94/ieice.html
- [4] C. Flavián, S. Ibáñez-Sánchez, and C. Orús, “The impact of virtual, augmented and mixed reality technologies on the customer experience,” *Journal of Business Research*, vol. 100, pp. 547–560, Jul. 2019, ISSN: 01482963. DOI: [10.1016/j.jbusres.2018.10.050](https://doi.org/10.1016/j.jbusres.2018.10.050)
- [5] W. D. Hoyer, M. Kroschke, B. Schmitt, K. Kraume, and V. Shankar, “Transforming the customer experience through new technologies,” *Journal of Interactive Marketing*, vol. 51, pp. 57–71, Aug. 2020, ISSN: 15206653. DOI: [10.1016/j.intmar.2020.04.001](https://doi.org/10.1016/j.intmar.2020.04.001)
- [6] P. A. Rauschnabel, R. Felix, C. Hinsch, H. Shahab, and F. Alt, “What is xr? towards a framework for augmented and virtual reality,” *Computers in Human Behavior*, vol. 133, Aug. 2022, ISSN: 07475632. DOI: [10.1016/j.chb.2022.107289](https://doi.org/10.1016/j.chb.2022.107289)
- [7] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, “Digital twin in manufacturing: A categorical literature review and classification,” in *IFAC-PapersOnLine*, vol. 51, Elsevier B.V., Jan. 2018, pp. 1016–1022. DOI: [10.1016/j.ifacol.2018.08.474](https://doi.org/10.1016/j.ifacol.2018.08.474)
- [8] F. Chiang, X. Shang, and L. Qiao, “Augmented reality in vocational training: A systematic review of research and applications,” *Computers in Human Behavior*, vol. 129, p. 107 125, Apr. 2022. DOI: [10.1016/j.chb.2021.107125](https://doi.org/10.1016/j.chb.2021.107125)
- [9] X. Han, Y. Chen, Q. Feng, and H. Luo, *Augmented reality in professional training: A review of the literature from 2001 to 2020*, Feb. 2022. DOI: [10.3390/app12031024](https://doi.org/10.3390/app12031024)
- [10] S. Machala, N. Chamier-Gliszczyński, and T. Królikowski, “Application of ar/vr technology in industry 4.0,” in *Procedia Computer Science*, vol. 207, Elsevier B.V., 2022, pp. 2984–2992. DOI: [10.1016/j.procs.2022.09.357](https://doi.org/10.1016/j.procs.2022.09.357)
- [11] A. Sarhan, M. Leray, T. Creusier, and J. Gao, “Augmented reality knowledge management for industrial transformation and innovation,” *Procedia CIRP*, vol. 128, pp. 19–24, 2024, 34th CIRP Design Conference, ISSN: 2212-8271. DOI: <https://doi.org/10.1016/j.procir.2024.06.004> [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2212827124006504>

- [12] G. Apostolopoulos, D. Andronas, N. Fourtakas, and S. Makris, “Operator training framework for hybrid environments: An augmented reality module using machine learning object recognition,” in *Procedia CIRP*, vol. 106, Elsevier B.V., 2022, pp. 102–107. DOI: [10.1016/j.procir.2022.02.162](https://doi.org/10.1016/j.procir.2022.02.162)
- [13] P. Barros, “Implementation of a mixed reality instructing system for assembly operations,” Master’s Thesis, Faculty of Engineering of the University of Porto, 2024.
- [14] C. R. Verdu, N. S. Macia, M. Strand, M. Holm, B. Schmidt, and J. Olsson, “Enhancing manual assembly training using mixed reality and virtual sensors,” in *Procedia CIRP*, vol. 126, Elsevier B.V., 2024, pp. 769–774. DOI: [10.1016/j.procir.2024.08.328](https://doi.org/10.1016/j.procir.2024.08.328)
- [15] S. Makris, G. Pintzos, L. Rentzos, and G. Chryssolouris, “Assembly support using ar technology based on automatic sequence generation,” *CIRP Annals - Manufacturing Technology*, vol. 62, pp. 9–12, 1 2013, ISSN: 00078506. DOI: [10.1016/j.cirp.2013.03.095](https://doi.org/10.1016/j.cirp.2013.03.095)
- [16] S. Makris, P. Karagiannis, S. Koukas, and A. S. Matthaiakis, “Augmented reality system for operator support in human–robot collaborative assembly,” *CIRP Annals - Manufacturing Technology*, vol. 65, pp. 61–64, 1 2016, ISSN: 17260604. DOI: [10.1016/j.cirp.2016.04.038](https://doi.org/10.1016/j.cirp.2016.04.038)
- [17] G. Michalos et al., “Seamless human robot collaborative assembly – an automotive case study,” *Mechatronics*, vol. 55, pp. 194–211, Nov. 2018, ISSN: 09574158. DOI: [10.1016/j.mechatronics.2018.08.006](https://doi.org/10.1016/j.mechatronics.2018.08.006)
- [18] C. M. Costa et al., “Modeling of video projectors in opengl for implementing a spatial augmented reality teaching system for assembly operations,” in *19th IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2019*, Institute of Electrical and Electronics Engineers Inc., Apr. 2019, ISBN: 9781728135588. DOI: [10.1109/ICARSC.2019.8733617](https://doi.org/10.1109/ICARSC.2019.8733617)
- [19] A. Blaga, C. Militaru, A. D. Mezei, and L. Tamas, “Augmented reality integration into mes for connected workers,” *Robotics and Computer-Integrated Manufacturing*, vol. 68, Apr. 2021, ISSN: 07365845. DOI: [10.1016/j.rcim.2020.102057](https://doi.org/10.1016/j.rcim.2020.102057)
- [20] K. B. Park, S. H. Choi, M. Kim, and J. Y. Lee, “Deep learning-based mobile augmented reality for task assistance using 3d spatial mapping and snapshot-based rgb-d data,” *Computers and Industrial Engineering*, vol. 146, Aug. 2020, ISSN: 03608352. DOI: [10.1016/j.cie.2020.106585](https://doi.org/10.1016/j.cie.2020.106585)
- [21] M. Fathi, I. Karlsson, G. Grahn, and A. Björnsson, “Unveiling the potential of mixed reality: Enhancing time measurement and operator support in manual assembly processes,” in *Procedia Computer Science*, vol. 232, Elsevier B.V., 2024, pp. 2670–2679. DOI: [10.1016/j.procs.2024.02.084](https://doi.org/10.1016/j.procs.2024.02.084)
- [22] M. L. Yuan, S. K. Ong, and A. Y. C. Nee, “Assembly guidance in augmented reality environments using a virtual interactive tool,” *International Journal of Production Research*, vol. 46, pp. 1745–1767, 7 Feb. 2008. DOI: [http://dx.doi.org/10.1080/00207540600972935](https://dx.doi.org/10.1080/00207540600972935) [Online]. Available: <http://hdl.handle.net/1721.1/7442>
- [23] S. H. Choi et al., “An integrated mixed reality system for safety-aware human-robot collaboration using deep learning and digital twin generation,” *Robotics and Computer-Integrated Manufacturing*, vol. 73, Feb. 2022, ISSN: 07365845. DOI: [10.1016/j.rcim.2021.102258](https://doi.org/10.1016/j.rcim.2021.102258)

- [24] N. Calzone, M. Sileo, R. Mozzillo, F. Pierri, and F. Caccavale, "Mixed reality platform supporting human-robot interaction," in *Lecture Notes in Mechanical Engineering*, Springer Science and Business Media Deutschland GmbH, 2023, pp. 1172–1182, ISBN: 9783031159275. DOI: [10.1007/978-3-031-15928-2_102](https://doi.org/10.1007/978-3-031-15928-2_102)
- [25] G. Carriero, N. Calzone, M. Sileo, F. Pierri, F. Caccavale, and R. Mozzillo, "Human-robot collaboration: An augmented reality toolkit for bi-directional interaction," *Applied Sciences (Switzerland)*, vol. 13, 20 Oct. 2023, ISSN: 20763417. DOI: [10.3390/app132011295](https://doi.org/10.3390/app132011295)
- [26] N. Kousi et al., "Digital twin for designing and reconfiguring human-robot collaborative assembly lines," *Applied Sciences (Switzerland)*, vol. 11, 10 May 2021, ISSN: 20763417. DOI: [10.3390/app11104620](https://doi.org/10.3390/app11104620)
- [27] K. Lalik and S. Flaga, "A real-time distance measurement system for a digital twin using mixed reality goggles," *Sensors*, vol. 21, 23 Dec. 2021, ISSN: 14248220. DOI: [10.3390/s21237870](https://doi.org/10.3390/s21237870)
- [28] D. Janecký, E. Kučera, O. Haffner, E. Výchlopečňová, and D. Rosinová, "Using mixed reality for control and monitoring of robot model based on robot operating system 2," *Electronics (Switzerland)*, vol. 13, 17 Sep. 2024, ISSN: 20799292. DOI: [10.3390/electronics13173554](https://doi.org/10.3390/electronics13173554)
- [29] S. Wang, R. Guo, H. Wang, Y. Ma, and Z. Zong, "Manufacture assembly fault detection method based on deep learning and mixed reality," in *Proceedings of the IEEE International Conference on Information and Automation*, IEEE Catalog Number: CFP18833-ART, IEEE, 2018, ISBN: 9781538680698.
- [30] E. Bottani et al., "Wearable and interactive mixed reality solutions for fault diagnosis and assistance in manufacturing systems: Implementation and testing in an aseptic bottling line," *Computers in Industry*, vol. 128, Jun. 2021, ISSN: 01663615. DOI: [10.1016/j.compind.2021.103429](https://doi.org/10.1016/j.compind.2021.103429)
- [31] T. V. Tom Warren, *Microsoft is discontinuing its HoloLens headsets*, Accessed: Jan. 11, 2025, 2024. [Online]. Available: <https://www.theverge.com/2024/10/1/24259369/microsoft-hololens-2-discontinuation-support>
- [32] Magic Leap, *Magic Leap 2*, Accessed: 21 July 2025. [Online]. Available: <https://www.magicleap.com/magic-leap-2>
- [33] J. Fingas, "Magic Leap's smaller, lighter second-gen AR glasses are now available," *Engadget*, 2022, Accessed: 21 July 2025. [Online]. Available: <https://www.engadget.com/magic-leap-2-ar-glasses-price-release-date-180041487.html>
- [34] Unity Technologies, *Unity Engine Official Website*, <https://unity.com/products/unity-engine>, Accessed: Jan. 11, 2025.
- [35] OGRE Team, *OGRE3D Official Website*, Accessed: Jan. 11, 2025. [Online]. Available: <https://www.ogre3d.org/>
- [36] Gazebo Team, *Gazebo Simulator Official Website*, Accessed: Jan. 11, 2025. [Online]. Available: <https://gazebo.org/>
- [37] Microsoft, *What is Mixed Reality Toolkit 2?* Accessed: June 13, 2025, 2022. [Online]. Available: <https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05>
- [38] Microsoft, *Mixed Reality Toolkit 3*, Accessed: June 13, 2025, 2023. [Online]. Available: <https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk3-overview/?view=mrtkunity-2022-05>

- [39] Google, *Google's ARCore on Google Developers*, Accessed: Jan. 11, 2025. [Online]. Available: <https://developers.google.com/ar>
- [40] Microsoft, *Dynamics 365 Guides documentation*, Accessed: 15 July 2024. [Online]. Available: <https://learn.microsoft.com/en-us/dynamics365/mixed-reality/guides/>
- [41] Microsoft, *Using the HoloLens Emulator*, Accessed: 30 July 2025. [Online]. Available: <https://learn.microsoft.com/en-us/windows/mixed-reality/develop/advanced-concepts/using-the-hololens-emulator>
- [42] Apple, *Capturing depth using the LiDAR camera*, Accessed: June 18, 2025. [Online]. Available: <https://developer.apple.com/documentation/avfoundation/capturing-depth-using-the-lidar-camera>
- [43] AR Code, *AR Code Object Capture*, Accessed: 15 July 2025. [Online]. Available: <https://ar-code.com/page/object-capture>
- [44] PTC, *Model Target Generator User Guide*, Accessed: 18 July 2025. [Online]. Available: <https://developer.vuforia.com/library/vuforia-engine/images-and-objects/model-targets/model-target-generator/model-target-generator-user-guide/>
- [45] PTC, *Model Targets*, Accessed: 19 July 2025. [Online]. Available: <https://developer.vuforia.com/library/vuforia-engine/images-and-objects/model-targets/model-targets/>
- [46] PTC, *Advanced Model Target Databases*, Accessed: 18 July 2025. [Online]. Available: <https://developer.vuforia.com/library/vuforia-engine/images-and-objects/model-targets/advanced-model-targets/advanced-model-target-databases/>
- [47] PTC, *Image Targets*, Accessed: 19 July 2025. [Online]. Available: <https://developer.vuforia.com/library/vuforia-engine/images-and-objects/image-targets/image-targets/>
- [48] SSSigun Studio, *Toony Workshop Tools FREE*, Unity Asset Store, Latest Release: Feb 25, 2025; Accessed: 3 June 2025. [Online]. Available: <https://assetstore.unity.com/packages/3d/props/toony-workshop-tools-free-311880>

Appendix A

Unity Project Scripts

This appendix presents the scripts implemented in the Unity project for training management and assembly processes.

A.1 Mode Manager Script

This script manages the training mode selection within the application. It allows users to choose between Virtual, Mixed, and Full Training modes and ensures that the selected mode persists across scenes in the Unity project.

```
1 using UnityEngine;
2 using UnityEngine.SceneManagement;
3
4 public class ModeManager : MonoBehaviour
5 {
6     public static ModeManager Instance { get; private set; }
7     public enum TrainingMode
8     {
9         None, // Default or unselected state
10        VirtualTraining,
11        MixedTraining,
12        FullTraining
13    }
14
15    public TrainingMode CurrentAssemblyMode = TrainingMode.None;
16
17    private void Awake()
18    {
19        if (Instance != null && Instance != this)
20        {
21            Destroy(gameObject);
22        }
23        else
```

```

24     {
25         Instance = this;
26         DontDestroyOnLoad(gameObject);
27     }
28 }
29 private void SetModeAndLoadMainScene(TrainingMode modeToLoad) //
    Changed parameter type to enum
30 {
31     // First, set the global mode variable
32     CurrentAssemblyMode = modeToLoad;
33     Debug.Log($"ModeManager: Mode set to: {CurrentAssemblyMode}");
34
35     // Then, load the main scene
36     if (CurrentAssemblyMode != TrainingMode.None)
37     {
38         SceneManager.LoadScene("MainScene");
39     }
40     else
41     {
42         Debug.LogError("ModeManager: No valid mode selected. Cannot
            load scene.");
43     }
44 }
45 public void OnVirtualTrainingButtonClicked()
46 {
47     SetModeAndLoadMainScene(ModeManager.TrainingMode.VirtualTraining);
48 }
49 public void OnMixedTrainingButtonClicked()
50 {
51     SetModeAndLoadMainScene(ModeManager.TrainingMode.MixedTraining);
52 }
53 public void OnFullTrainingButtonClicked()
54 {
55     SetModeAndLoadMainScene(ModeManager.TrainingMode.FullTraining);
56 }
57 }

```

Listing A.1: ModeManager script for managing training modes

A.2 Assembly Logic Scripts

This section presents scripts responsible for managing the assembly process, including tracking object placement, guiding user interactions, and providing feedback during assembly tasks.

```
1 using UnityEngine;
```

```

2 using Vuforia;
3 using System.Collections.Generic; // Required for List<ScrewInteractive>
4 using MixedReality.Toolkit.SpatialManipulation; // Required for
   SceneManager
5
6 public class AssemblyManager : MonoBehaviour
7 {
8     // -- Assembly Mode --
9     public ModeManager.TrainingMode assemblyMode = ModeManager.
        TrainingMode.None; // The current training mode, set by the
        ModeManager
10
11     // -- Model Tracking --
12
13     private ModelTargetBehaviour doorModelTargetBehaviour; // The
        ModelTargetBehaviour component for the door model target
14     private ModelTargetBehaviour mirrorModelTargetBehaviour; // The
        ModelTargetBehaviour component for the mirror model target
15
16     public bool isDoorTracked = false; // Internal state to track if the
        Model Target is currently being tracked
17     public bool isMirrorTracked = false; // Internal state to track if the
        mirror Model Target is currently being tracked
18
19     // --- Internal State Management ---
20
21     public enum AssemblyPhase
22     {
23         StartUp, // Initial state before any assembly begins,
            waits for door to be tracked
24         MirrorAssembly, // Waits for user to confirm mirror placement
25         MirrorAssemblyCheck, // Checks mirror placement
26         ScrewAssembly, // Placing screws
27         Complete // All assembly steps are finished.
28     }
29
30     public AssemblyPhase currentPhase = AssemblyPhase.StartUp; // The
        current phase of the assembly process, starting with Door Setup
31
32     // --- Public Variables to Assign in Inspector ---
33
34     public TextDisplay assemblyDisplay; // Text display for assembly
        instructions and feedback
35

```

```
36 public GameObject screwsObject; // The GameObject that contains the
    screws and guides.
37 public TextDisplay screwsDisplay; // Text display for screw count
    and feedback
38
39 public GameObject quarterGlassObject; // The quarter glass object
40
41 // --- Assembly Parts ---
42
43 private List<ScrewInteractive> individualScrews = new List<
    ScrewInteractive>();
44 private AssemblyCheck mirrorAssembly; // The script that manages the
    mirror assembly logic.
45
46 // --- Internal State Variables ---
47 private bool isMirrorPlacedCorrectly = false; // Tracks if the mirror
    is placed correctly
48
49 private void Awake()
50 {
51     if (ModeManager.Instance != null)
52     {
53         // Ensure the assembly mode is set before proceeding
54         assemblyMode = ModeManager.Instance.CurrentAssemblyMode;
55     }
56
57     // Get door ModelTargetBehaviour component directly from parent
    GameObject
58     doorModelTargetBehaviour = GetComponentInParent<
    ModelTargetBehaviour>();
59
60     //Get mirror ModelTargetBehaviour component directly from child
    GameObject
61     mirrorModelTargetBehaviour = GetComponentInChildren<
    ModelTargetBehaviour>();
62
63     if (doorModelTargetBehaviour == null || mirrorModelTargetBehaviour
        == null)
64     {
65         Debug.LogError("AssemblyManager: ModelTargetBehaviour not
            found in Full Training mode. Disabling script.", this);
66         enabled = false;
67         return;
68     }
69
```

```
70 // Find mirror assembly script and assign assembly manager
71 mirrorAssembly = GetComponentInChildren<AssemblyCheck>();
72 if (mirrorAssembly == null)
73 {
74     Debug.LogError("AssemblyManager: AssemblyCheck script not
75         found in children. Please ensure it is attached to a child
76         GameObject.", this);
77     enabled = false;
78     return;
79 }
80 mirrorAssembly.assemblyManager = this; // Give the mirror assembly
81 script a reference to this manager
82
83 // Find all ScrewInteractive scripts in children of screwsObject
84 if (screwsObject != null)
85 {
86     individualScrews.AddRange(screwsObject.GetComponentsInChildren
87         <ScrewInteractive>());
88
89     if (individualScrews.Count == 0)
90     {
91         Debug.LogWarning("AssemblyManager: No ScrewInteractive
92             scripts found as children of screwsObject. Screw phase
93             will not function.");
94     }
95
96     // Now, iterate through the found screws and give them a
97     reference back to this manager.
98     foreach (ScrewInteractive screw in individualScrews)
99     {
100         screw.assemblyManager = this; // Give each screw a
101         reference back to this manager
102     }
103 }
104 else
105 {
106     Debug.LogError("AssemblyManager: 'Screws Object' is not
107         assigned. Please assign it in the Inspector.", this);
108     enabled = false;
109     return;
110 }
111
112 if (assemblyDisplay == null)
113 {
```

```

106         Debug.LogError("AssemblyManager: 'Assembly Display' (
            TextMeshPro) is not assigned. Please assign it in the
            Inspector.", this);
107         enabled = false;
108         return;
109     }
110
111     if (screwsDisplay == null)
112     {
113         Debug.LogError("AssemblyManager: 'Screws Display' (TextMeshPro
            for screw count) is not assigned. Please assign it in the
            Inspector.", this);
114         enabled = false;
115         return;
116     }
117
118     if (individualScrews.Count == 0)
119     {
120         Debug.LogError("AssemblyManager: 'Individual Screws' list is
            empty. Please assign screw GameObjects to the list in the
            Inspector.", this);
121         enabled = false;
122         return;
123     }
124
125 }
126
127 private void Start()
128 {
129     switch (assemblyMode)
130     {
131         case ModeManager.TrainingMode.VirtualTraining:
132
133             // Disable Vuforia entirely in VirtualTraining mode
134             VuforiaBehaviour.Instance.enabled = false;
135
136             if (VuforiaApplication.Instance.IsRunning)
137                 VuforiaApplication.Instance.Deinit(); // fully
                    deinitializes Vuforia
138
139             isDoorTracked = true; // Assume door is tracked in
                    VirtualOnly mode
140             isMirrorTracked = true; // Assume mirror is tracked in
                    VirtualOnly mode
141

```



```
142         // Turn off door tracking
143         StopObjectTracking (doorModelTargetBehaviour);
144
145         // Turn off mirror tracking
146         StopObjectTracking (mirrorModelTargetBehaviour);
147
148         // Make door and mirror visible
149         mirrorAssembly.gameObject.GetComponent<Renderer>().enabled
150             = true;
151         this.gameObject.GetComponent<Renderer>().enabled = true;
152         break;
153
154     case ModeManager.TrainingMode.MixedTraining:
155
156         isDoorTracked = false; // Initially not tracked in
157             MixedTraining mode
158         isMirrorTracked = true; // Assume mirror is tracked in
159             MixedTraining mode
160
161         // Enable door Model Target tracking
162         TurnModelTargetOnandOff (doorModelTargetBehaviour, true);
163
164         // Turn off mirror tracking
165         StopObjectTracking (mirrorModelTargetBehaviour);
166
167         // Disable quarter glass in MixedTraining mode
168         quarterGlassObject.SetActive(false);
169
170         break;
171
172     case ModeManager.TrainingMode.FullTraining:
173
174         isDoorTracked = false; // Initially not tracked in
175             FullTraining mode
176         isMirrorTracked = false; // Initially not tracked in
177             FullTraining mode
178
179         // Enable door Model Target tracking
180         TurnModelTargetOnandOff (doorModelTargetBehaviour, true);
181
182         // Disable mirror Model Target tracking for now and stop
183             mirror manipulation
```

```

179         TurnModelTargetOnandOff(mirrorModelTargetBehaviour, false)
            ; // Start with mirror model target disabled until user
              confirms placement
180         mirrorAssembly.gameObject.GetComponent<ObjectManipulator
            >().enabled = false; // Disable mirror manipulation
              script
181
182         quarterGlassObject.SetActive(false); // Disable quarter
              glass in FullTraining mode
183
184         break;
185     }
186
187     SetObjectState(screwsObject, false); // Hide screws and guides at
              start
188 }
189
190 void Update()
191 {
192     Transform screw1 = individualScrews[0].transform;
193     Transform screw2 = individualScrews[1].transform;
194     float distance = Vector3.Distance(screw1.position, screw2.position
        );
195     Debug.Log($"Distance between screw 1 and 2: {distance:F4} meters")
        ;
196
197     // ----- Visibility management -----
198
199     if (assemblyMode == ModeManager.TrainingMode.MixedTraining)
200     {
201         // Continuously make door transparent in MixedTraining mode
202         MakeInvisible(this.gameObject); // Make the door assembly
              hologram transparent
203     }
204     if (assemblyMode == ModeManager.TrainingMode.FullTraining)
205     {
206         // Continuously make door and mirror transparent in
              FullTraining mode
207         MakeInvisible(this.gameObject); // Door
208         MakeInvisible(mirrorAssembly.gameObject); // Mirror
209     }
210
211     // --- Core Assembly State Machine ---
212     switch (currentPhase)
213     {

```

```
214
215     case AssemblyPhase.StartUp:
216         if (isDoorTracked)
217         {
218             currentPhase = AssemblyPhase.MirrorAssembly; //
219                 Transition to mirror assembly phase
220             if (assemblyMode == ModeManager.TrainingMode.
221                 FullTraining)
222                 mirrorAssembly.ShowMirrorAssemblyInstruction();
223         }
224         break;
225
226     case AssemblyPhase.MirrorAssembly:
227
228         if (assemblyMode != ModeManager.TrainingMode.FullTraining)
229         {
230             currentPhase = AssemblyPhase.MirrorAssemblyCheck; //
231                 Transition to Mirror Detection phase
232         }
233         break;
234
235     case AssemblyPhase.MirrorAssemblyCheck:
236
237         if (isMirrorTracked)
238         {
239             if (assemblyMode == ModeManager.TrainingMode.
240                 FullTraining)
241             {
242                 mirrorAssembly.CheckMirrorPlacement();
243             }
244
245             if (isMirrorPlacedCorrectly)
246             {
247                 currentPhase = AssemblyPhase.ScrewAssembly; //
248                     Transition to screw assembly phase
249
250                 if (assemblyMode == ModeManager.TrainingMode.
251                     FullTraining)
252                 {
253                     StopObjectTracking(mirrorModelTargetBehaviour)
254                     ; // Freeze mirror in place
255                 }
256             }
257         }
258     }
```

```

250         TurnModelTargetOnandOff(
                doorModelTargetBehaviour, true); // Enable
                door model target to resume tracking
251     }
252
253     SetObjectState(screwsObject, true); // Show screws
                and guides
254     assemblyDisplay.ShowTemporaryText("Mirror placed
                correctly", 3.0f, Color.green);
255 }
256 else if (assemblyMode == ModeManager.TrainingMode.
                FullTraining)
257 {
258     currentPhase = AssemblyPhase.MirrorAssembly; //
                Retry mirror placement
259
260     TurnModelTargetOnandOff(mirrorModelTargetBehaviour
                , false); // Disable mirror model target to
                pause tracking
261     TurnModelTargetOnandOff(doorModelTargetBehaviour,
                true); // Enable door model target to resume
                tracking
262     assemblyDisplay.ShowTemporaryText("Place mirror
                correctly", 2.0f, Color.red);
263
264     mirrorAssembly.ShowMirrorAssemblyInstruction();
                // Show instructions again
265
266 }
267 }
268
269 break;
270
271 case AssemblyPhase.ScrewAssembly:
272
273     //mirrorAssembly.gameObject.GetComponent<ObjectManipulator
                >().enabled = false; // Disable mirror manipulation
                script to prevent further movement
274
275     if (!isMirrorPlacedCorrectly)
276     {
277         currentPhase = AssemblyPhase.MirrorAssembly; // Go
                back to mirror assembly phase if mirror is moved
278     }
279

```

```

280         break;
281
282     case AssemblyPhase.Complete:
283
284         assemblyDisplay.SetPersistentText("Assembly Complete",
285             Color.green);
286
287         if (!isMirrorPlacedCorrectly)
288         {
289             currentPhase = AssemblyPhase.MirrorAssembly; // Go
290                 back to mirror assembly phase if mirror is moved
291         }
292         if (individualScrews.Exists(screw => !screw.
293             IsScrewInserted()))
294         {
295             currentPhase = AssemblyPhase.ScrewAssembly; // Go back
296                 to screw assembly phase if any screw is removed
297         }
298
299         break;
300     }
301 }
302
303 // -- Methods to update assembly status --
304 /// <summary>
305 /// Method for MirrorAssembly script to call to update the mirror's
306 position and rotation status.
307 /// </summary>
308 public void UpdateMirrorAssemblyStatus(bool isAssemblyCorrect)
309 {
310
311     Debug.Log($"AssemblyManager: Mirror assembly status updated. Is
312         assembly correct? {isAssemblyCorrect}");
313     isMirrorPlacedCorrectly = isAssemblyCorrect;
314 }
315 public void ConfirmMirrorPlacement()
316 {
317     if (currentPhase == AssemblyPhase.MirrorAssembly)
318     {
319         Debug.Log("AssemblyManager: User confirmed mirror placement.")
320             ;
321
322         TurnModelTargetOnandOff(doorModelTargetBehaviour, false); //
323             Disable door model target to stop tracking

```

```

316         TurnModelTargetOnandOff(mirrorModelTargetBehaviour, true); //
           Enable mirror model target to start tracking
317
318         // Transition to mirror check phase
319         currentPhase = AssemblyPhase.MirrorAssemblyCheck;
320     }
321 }
322
323 /// <summary>
324 /// This method is called by each ScrewInteractive script whenever a
           screw's inserted state changes.
325 /// It updates the total count of inserted screws and refreshes the UI
           feedback.
326 /// </summary>
327 /// <param name="screwName">The name of the screw that changed state (
           for logging).</param>
328 /// <param name="inserted">True if the screw was inserted, false if
           removed.</param>
329
330 public void UpdateScrewAssemblyStatus(string screwName, bool inserted)
331 {
332     int insertedScrewCount = 0;
333     foreach (ScrewInteractive screw in individualScrews)
334     {
335         if (screw.IsScrewInserted())
336         {
337             insertedScrewCount++;
338         }
339     }
340
341     if (currentPhase == AssemblyPhase.ScrewAssembly)
342     {
343
344         if (insertedScrewCount == 0) // If no screws are inserted and
           we are in Screw phase
345         {
346             screwsDisplay.SetPersistentText("Insert screws", Color.red
           ); // Show red feedback
347         }
348         else if (insertedScrewCount < individualScrews.Count) // If
           not all screws are inserted
349         {
350             screwsDisplay.SetPersistentText($"{insertedScrewCount} / {
           individualScrews.Count} screws inserted", Color.yellow)
           ;

```

```
351
352     }
353     else // If all screws are inserted
354     {
355         screwsDisplay.ShowTemporaryText("All screws inserted", 2.0
356             f, Color.green); // Show success message
357
358         // Transition to complete phase
359         currentPhase = AssemblyPhase.Complete;
360         Debug.Log("Assembly Complete");
361     }
362     else
363     {
364         if (insertedScrewCount < individualScrews.Count &&
365             currentPhase == AssemblyPhase.Complete)
366         {
367             currentPhase = AssemblyPhase.ScrewAssembly;
368         }
369     }
370
371     /// <summary>
372     /// Makes the GameObject invisible by disabling its Renderer.
373     /// This is used to make objects transparent or hidden in the scene
374     /// without deactivating them.
375     /// </summary>
376     private void MakeInvisible(GameObject gameObject)
377     {
378         Renderer renderer = gameObject.GetComponent<Renderer>();
379         if (renderer != null)
380             renderer.enabled = false; // Disable rendering to make it
381             invisible
382         //Debug.Log($" {gameObject.name} turned transparent");
383     }
384     private void SetObjectState(GameObject targetObject, bool isActive)
385     {
386         if (targetObject != null)
387         {
388             targetObject.SetActive(isActive);
389         }
390     }
```

```
391 private void TurnModelTargetOnandOff (ModelTargetBehaviour modelTarget,
392     bool turnOn)
393 {
394     if (modelTarget != null)
395     {
396         Debug.Log($"AssemblyManager: ModelTargetBehaviour for {
397             modelTarget.gameObject.name} will be turned {(turnOn ? "on"
398                 : "off")}");
399         modelTarget.enabled = turnOn;
400
401         if (turnOn)
402         {
403             if (modelTarget == doorModelTargetBehaviour)
404                 assemblyDisplay.SetPersistentText("Look at door",
405                     Color.red); // Ask user to look at door
406             if (modelTarget == mirrorModelTargetBehaviour)
407                 assemblyDisplay.SetPersistentText("Look at mirror",
408                     Color.red); // Ask user to look at mirror
409         }
410     }
411     else
412     {
413         Debug.LogWarning("AssemblyManager: Attempted to turn on/off a
414             null ModelTargetBehaviour reference.");
415     }
416 }
417
418 private void StopObjectTracking (ModelTargetBehaviour
419     modelTargetBehaviour)
420 {
421     if (modelTargetBehaviour != null)
422     {
423         if (modelTargetBehaviour == mirrorModelTargetBehaviour)
424         {
425             mirrorAssembly.transform.SetParent(this.gameObject.
426                 transform, true); // Unparent the mirror representation
427                 to "freeze" it in world space.
428             SetObjectState(modelTargetBehaviour.gameObject, false);
429             // Set mirror model target as inactive to prevent
430             further tracking
431         }
432         else if (modelTargetBehaviour == doorModelTargetBehaviour)
433         {
434             // Freeze door in place by unparenting it from any model
435             target and disabling tracking
```



```
424         this.gameObject.transform.SetParent(null, true); // Set
           door assembly as a child of this GameObject
425         SetObjectState(modelTargetBehaviour.gameObject, false);
           // Disable door model target to prevent further
           tracking
426     }
427     else
428     {
429         Debug.LogWarning("AssemblyManager: Attempted to stop an
           unrecognized Model Target tracking.");
430     }
431 }
432
433 }
434 public void SetDoorAsTracked()
435 {
436     isDoorTracked = true;
437     assemblyDisplay.ClearAllText(); // Clear any previous messages
438
439     // Parent the door assembly to the model target to enable tracking
440     this.gameObject.transform.SetParent(doorModelTargetBehaviour.
           transform, true); // Set this GameObject as a child of the door
           model target
441
442     Debug.Log("AssemblyManager: Door is now tracked.");
443 }
444
445 public void SetDoorAsUntracked()
446 {
447     isDoorTracked = false;
448     assemblyDisplay.SetPersistentText("Look at door", Color.red); //
           Show error message
449
450     // Freeze door in place by unparenting it from any model target
451     this.gameObject.transform.SetParent(null, true); // Set door
           assembly as a child of this GameObject
452
453     Debug.Log("AssemblyManager: Door is now untracked.");
454 }
455
456 public void SetMirrorAsTracked()
457 {
458     isMirrorTracked = true;
459     assemblyDisplay.ClearAllText(); // Clear any previous messages
460 }
```

```

461     Debug.Log("AssemblyManager: Mirror is now tracked.");
462 }
463
464 public void SetMirrorAsUntracked()
465 {
466     isMirrorTracked = false;
467     if (currentPhase == AssemblyPhase.MirrorAssemblyCheck)
468     {
469         assemblyDisplay.SetPersistentText("Look at mirror", Color.red)
470         ; // Show error message
471     }
472
473     Debug.Log("AssemblyManager: Mirror is now untracked.");
474 }
475 }

```

Listing A.2: AssemblyManager script for managing the assembly process and feedback

```

1
2 using UnityEngine;
3 using System.Collections;
4 using MixedReality.Toolkit.UX;
5 using MixedReality.Toolkit.SpatialManipulation;
6
7 public class AssemblyCheck : MonoBehaviour
8 {
9
10     public AssemblyManager assemblyManager;
11     public GameObject placeObject; // The guide object to compare against
12     public GameObject mirrorDialogObject; // Dialog to wait for user input
13         that they understand the assembly instructions
14     public ObjectManipulator objectManipulator; // ObjectManipulator
15         component for enabling/disabling manipulation
16
17     private TextDisplay textDisplay; // TextDisplay component for showing
18         messages
19     private StaticDialog mirrorDialog; // Subclass of Dialog component
20         that doesn't follow user
21
22     public float placeAnimationOffset = 0.1f; // Offset for the placement
23         animation
24
25     private Coroutine mirrorPlaceLoopCoroutine;
26
27     // --- Tolerance settings for accuracy ---

```

```
23     public float positionTolerance = 0.10f; // Meters (e.g., 5 cm)
24     public float rotationToleranceDegrees = 7.0f; // Degrees
25
26     private bool prevPositionCorrect = false, prevRotationCorrect = false;
27     private bool positionCorrect = false, rotationCorrect = false;
28
29     private Vector3 endPos;
30
31     private void Awake()
32     {
33         // Get the TextDisplay component from any child GameObject
34         textDisplay = placeObject.GetComponentInChildren<TextDisplay>();
35
36         // Always good to check if it was found
37         if (textDisplay == null)
38         {
39             Debug.LogError($"ParentScript: TextDisplay component not found
40                             on any child of {gameObject.name}. Please ensure it's
41                             attached to a child GameObject.");
42             enabled = false;
43             return;
44         }
45
46         mirrorDialog = mirrorDialogObject.GetComponent<StaticDialog>();
47
48         if (mirrorDialog == null)
49         {
50             Debug.LogError($"ParentScript: StaticDialog component not
51                             found on any child of {gameObject.name}. Please ensure it's
52                             attached to a child GameObject.");
53             enabled = false;
54             return;
55         }
56
57         objectManipulator = GetComponent<ObjectManipulator>();
58
59     }
60     private void Start()
61     {
62         if (assemblyManager == null || placeObject == null)
63         {
64             Debug.LogError("AssemblyCheck: Missing required references.");
65             enabled = false;
66             return;
67         }
68     }
```

```

64
65     if (assemblyManager.assemblyMode != ModeManager.TrainingMode.
66         FullTraining)
67     {
68         DeactivateDialog(null);
69     }
70
71 }
72
73 private void Update()
74 {
75     if (assemblyManager.assemblyMode == ModeManager.TrainingMode.
76         VirtualTraining || assemblyManager.assemblyMode == ModeManager.
77         TrainingMode.MixedTraining)
78     {
79         CheckMirrorPlacement();
80     }
81
82 public void CheckMirrorPlacement()
83 {
84     Vector3 currentPosition = transform.position;
85     Quaternion currentRotation = transform.rotation;
86
87     // --- Get Target Position/Rotation (from the guide object)
88     Vector3 targetLocalPosition = placeObject.transform.position;
89     Quaternion targetLocalRotation = placeObject.transform.rotation;
90
91     Debug.Log($"Current pos: {currentPosition} -> Target pos {
92         targetLocalPosition}");
93
94     // --- Compare Positions ---
95     positionCorrect = Vector3.Distance(currentPosition,
96         targetLocalPosition) <= positionTolerance;
97
98     // --- Compare Rotations ---
99     float angleDiff = Quaternion.Angle(currentRotation,
100         targetLocalRotation);
101     rotationCorrect = angleDiff <= rotationToleranceDegrees;
102
103     if (prevPositionCorrect != positionCorrect || prevRotationCorrect
104         != rotationCorrect) // If there is a position change
105     {
106         // --- Debugging Information ---

```

```
102     Debug.Log($"Current pos: {currentPosition} -> Target pos {
        targetLocalPosition}");
103     Debug.Log($"Current rot: {currentRotation.eulerAngles} ->
        Target rot {targetLocalRotation.eulerAngles}");
104
105     // --- Update the AssemblyManager ---
106     assemblyManager.UpdateMirrorAssemblyStatus(positionCorrect &&
        rotationCorrect);
107
108     if (positionCorrect && rotationCorrect)
109     {
110         placeObject.SetActive(false); // Hide the guide object
        when assembly is correct
111
112         if (assemblyManager.assemblyMode != ModeManager.
            TrainingMode.FullTraining)
113         {
114             if (objectManipulator != null)
115             {
116                 objectManipulator.enabled = false;
117             }
118
119             // Clip both position AND rotation in World Space
120             this.gameObject.transform.position = placeObject.
                transform.position;
121             this.gameObject.transform.rotation = placeObject.
                transform.rotation;
122         }
123     }
124     else // If assembly is incorrect
125     {
126         placeObject.gameObject.SetActive(true); // Show the guide
        object when assembly is incorrect
127         string feedback = "";
128         if (!positionCorrect) feedback += "Position Off ";
129         if (!rotationCorrect) feedback += "Rotation Off ";
130
131         Debug.Log($"MirrorAssembly: {feedback}");
132
133         if (assemblyManager.assemblyMode != ModeManager.
            TrainingMode.FullTraining)
134         {
135             textDisplay.SetPersistentText(feedback, Color.red); //
            Show error message
136
```

```

137         }
138         else // Full training mode
139         {
140             textDisplay.ClearAllText(); // Clear persistent text
141             textDisplay.ShowTemporaryText(feedback, 2.0f, Color.
                red); // Show error message
142         }
143     }
144     prevPositionCorrect = positionCorrect;
145     prevRotationCorrect = rotationCorrect;
146
147 }
148
149 private void StartMirrorDisplayLoop()
150 {
151     if (mirrorPlaceLoopCoroutine == null)
152     {
153         mirrorPlaceLoopCoroutine = StartCoroutine(
            MirrorDisplayCoroutine());
154     }
155 }
156 private IEnumerator MirrorDisplayCoroutine()
157 {
158     if (placeObject == null)
159     {
160         Debug.LogWarning("Place Object is not assigned. Please assign
            it in the Inspector.");
161         yield break;
162     }
163
164     // Consistent animation duration and loop times.
165     float animationDuration = 1f;
166     float visibleTime = 3f;
167     float hiddenTime = 1f;
168
169     endPos = placeObject.transform.localPosition;
170
171     while (true)
172     {
173         placeObject.SetActive(true);
174
175         Vector3 startPos = endPos + (Vector3.forward *
            placeAnimationOffset);
176
177         placeObject.transform.localPosition = startPos;

```

```
178
179     float elapsed = 0f;
180     while (elapsed < animationDuration)
181     {
182         placeObject.transform.localPosition = Vector3.Lerp(
183             startPos, endPos, elapsed / animationDuration);
184         elapsed += Time.deltaTime;
185         yield return null;
186     }
187
188     placeObject.transform.localPosition = endPos;
189
190     yield return new WaitForSeconds(visibleTime -
191         animationDuration);
192
193     placeObject.SetActive(false);
194
195     yield return new WaitForSeconds(hiddenTime);
196 }
197
198 private void StopMirrorDisplayLoop()
199 {
200     if (mirrorPlaceLoopCoroutine != null)
201     {
202         // Put place object back to original position and hide it
203
204         placeObject.SetActive(false); // Hide the guide object when
205             stopping the loop
206         StopCoroutine(mirrorPlaceLoopCoroutine);
207         placeObject.transform.localPosition = endPos;
208         mirrorPlaceLoopCoroutine = null;
209     }
210 }
211
212 public void ShowMirrorAssemblyInstruction()
213 {
214     StartMirrorDisplayLoop();
215     MirrorAnimationDialog();
216 }
217
218 public void HideMirrorAssemblyInstruction()
219 {
220     StopMirrorDisplayLoop();
221 }
```

```
220
221 // Dialog to dismiss animation
222 private void MirrorAnimationDialog()
223 {
224     ActivateDialog();
225
226     mirrorDialog.SetHeader("Confirm if you understand the assembly
227                             instruction");
228     mirrorDialog.SetNeutral("Understood", MirrorAssembledDialog);
229     mirrorDialog.Show();
230
231     Debug.Log("Mirror Assembly: Mirror animation dialog activated.");
232 }
233
234 // Dialog to confirm mirror placement
235 private void MirrorAssembledDialog(DialogButtonEventArgs args)
236 {
237     StopMirrorDisplayLoop();
238     ActivateDialog();
239     mirrorDialog.SetHeader("Confirm when the mirror is assembled
240                             correctly");
241
242     mirrorDialog.SetNeutral("Done", (args) => {
243
244         assemblyManager.ConfirmMirrorPlacement();
245         DeactivateDialog(args);
246
247     });
248     mirrorDialog.Show();
249
250     Debug.Log("Mirror Assembly: Mirror assemble confirmation dialog
251               activated.");
252 }
253
254 private void DeactivateDialog(DialogButtonEventArgs args)
255 {
256     Debug.Log("Deactivating mirror dialog...");
257
258     if (mirrorDialogObject != null)
259     {
260         mirrorDialogObject.SetActive(false);
261     }
```



```
262
263     }
264
265     private void ActivateDialog()
266     {
267         Debug.Log("Activating mirror dialog...");
268
269         if (mirrorDialogObject != null)
270         {
271             mirrorDialogObject.SetActive(true);
272         }
273     }
274
275
276 }
```

Listing A.3: MirrorAssembly script for managing mirror placement and validation

```
1
2 using UnityEngine;
3 using MixedReality.Toolkit;
4
5 public class ScrewInteractive : MonoBehaviour
6 {
7
8     private StatefulInteractable interactable;
9     public AssemblyManager assemblyManager; // Reference to the
10                                             AssemblyCheck manager for UI feedback
11
12     public GameObject screwGuide;
13
14     private Vector3 initialLocalPosition; // Stores the screw's starting
15                                             position
16     private bool isInserted = false; // Tracks if the screw is currently "
17                                             inserted"
18
19     [Tooltip("The amount to shift the screw inwards when inserted.")]
20     public Vector3 zShift = new Vector3(0, 0, 0.1f); // The amount to
21                                             shift the screw inwards when inserted
22
23     void Awake()
24     {
25         // Get the StatefulInteractable component attached to this
26         // GameObject
27         interactable = GetComponent<StatefulInteractable>();
28         if (interactable == null)
```

```

24     {
25         Debug.LogError($"ScrewInteractive: StatefulInteractable
           component missing on {gameObject.name}. Disabling script.")
           ;
26         enabled = false;
27         return;
28     }
29     initialLocalPosition = transform.localPosition;
30
31     // Ensure the AssemblyManager is assigned in the inspector or find
           it in the scene
32     if (assemblyManager == null)
33     {
34         assemblyManager = FindFirstObjectByType<AssemblyManager>(); //
           Try to find the AssemblyManager in the scene if not
           assigned in the inspector
35         if (assemblyManager == null)
36         {
37             Debug.LogError("ScrewInteractive: AssemblyManager not
           found in the scene. Please assign it in the inspector
           or ensure it's present.");
38             enabled = false;
39         }
40     }
41
42 }
43
44 void OnEnable()
45 {
46     assemblyManager.UpdateScrewAssemblyStatus(gameObject.name,
           isInserted); // Update the assembly manager with the initial
           state of the screw
47     // Subscribe to the OnClicked event from the StatefulInteractable.
           // This event triggers when a user performs a "select" action on
           the screw.
48     interactable.OnClicked.AddListener(HandleScrewClicked);
49 }
50
51
52 void OnDisable()
53 {
54     // Unsubscribe to prevent memory leaks
55     interactable.OnClicked.RemoveListener(HandleScrewClicked);
56 }
57
58 /// <summary>

```

```
59     /// This method is called by the StatefulInteractable when the screw
        is 'clicked'.
60     /// </summary>
61     private void HandleScrewClicked()
62     {
63         Debug.Log($"Screw {gameObject.name} clicked.");
64
65         ToggleScrewPosition(); // Put screw in correct position
66
67     }
68
69     /// <summary>
70     /// Toggles the screw's visual position between "inserted" and "
        removed" states.
71     /// </summary>
72     private void ToggleScrewPosition()
73     {
74         if (!isInserted)
75         {
76             // Move the screw inwards.
77             transform.localPosition = initialLocalPosition + zShift;
78             isInserted = true;
79             Debug.Log($"Screw {gameObject.name} inserted.");
80
81             // Turn Screw Guide off
82             screwGuide.SetActive(false);
83         }
84         else
85         {
86             // Move the screw back to its initial, "removed" position.
87             transform.localPosition = initialLocalPosition;
88             isInserted = false;
89             Debug.Log($"Screw {gameObject.name} removed.");
90
91             // Turn Screw Guide on again
92             screwGuide.SetActive(true);
93         }
94
95         // Inform the DoorAssemblyCheck manager about the screw's new
            state.
96         // This allows the manager to update the UI feedback (e.g., "X out
            of Y screws inserted").
97         if (assemblyManager != null)
98         {
```

```

99         assemblyManager.UpdateScrewAssemblyStatus(gameObject.name,
100             isInserted);
101     }
102 }
103
104 /// <summary>
105 /// Method to check if screw is currently inserted.
106 /// </summary>
107 public bool IsScrewInserted()
108 {
109     return isInserted;
110 }
111 }

```

Listing A.4: ScrewInteractive script for interactive manipulation and placement of screw objects in the Unity scene

A.3 Feedback Helper Scripts

These scripts handle user interface elements such as text component and dialogs, ensuring that feedback is clearly presented during the training process.

```

1
2 using UnityEngine;
3
4 public class Billboard : MonoBehaviour
5 {
6     void Update()
7     {
8         if (Camera.main != null)
9         {
10             transform.LookAt(Camera.main.transform);
11             transform.Rotate(0, 180, 0); // Correct for the text being
12                 backwards
13         }
14     }
15 }

```

Listing A.5: Billboard script used to face text elements toward the user

```

1
2 using UnityEngine;
3 using TMPro;
4 using System.Collections;

```

```
5
6 public class TextDisplay : MonoBehaviour
7 {
8     public enum TextComponentType
9     {
10         TextMeshPro_3D,
11         TextMeshPro_UGUI
12     }
13
14     public TextComponentType selectedTextType;
15
16     private TextMeshPro _displayTextMesh3D;
17     private TextMeshProUGUI _displayTextMeshUGUI;
18
19     private TMP_Text _activeTextMesh;
20     private Color _initialTextColor; // Stores the color of the text
21                                     // component when the script starts
22
23     private Coroutine currentDisplayCoroutine;
24
25     void Awake()
26     {
27         if (selectedTextType == TextComponentType.TextMeshPro_3D)
28         {
29             _displayTextMesh3D = GetComponent<TextMeshPro>();
30             _activeTextMesh = _displayTextMesh3D;
31         }
32         else
33         {
34             _displayTextMeshUGUI = GetComponent<TextMeshProUGUI>();
35             _activeTextMesh = _displayTextMeshUGUI;
36         }
37
38         if (_activeTextMesh == null)
39         {
40             Debug.LogError($"TemporaryTextDisplay: Expected a {
41                             selectedTextType} component on this GameObject, but none
42                             was found. Please ensure the correct TextMeshPro component
43                             is attached.");
44             enabled = false;
45         }
46         else
47         {
48             _initialTextColor = _activeTextMesh.color; // Store the
49                                                         // current color of the text
50         }
51     }
52 }
```

```

45     _activeTextMesh.text = ""; // Clear text initially
46 }
47 }
48
49 /// <summary>
50 /// Sets the text to a message for a specified duration, then clears
51 /// it.
52 /// If another message is already being displayed, the previous one is
53 /// cancelled.
54 /// </summary>
55 /// <param name="message">The text message to display.</param>
56 /// <param name="duration">How long to display the message in seconds
57 /// .</param>
58 /// <param name="color">Optional: The color of the text. If null, uses
59 /// the text's initial color.</param>
60 public void ShowTemporaryText(string message, float duration, Color?
61     color = null)
62 {
63     if (_activeTextMesh == null) return;
64
65     // Stop any existing coroutine to prevent overlapping messages
66     if (currentDisplayCoroutine != null)
67     {
68         StopCoroutine(currentDisplayCoroutine);
69     }
70
71     // Apply the specified color, or fall back to the initial color if
72     // none provided
73     _activeTextMesh.color = color ?? _initialTextColor;
74
75     // Start the new coroutine
76     currentDisplayCoroutine = StartCoroutine(DisplayAndClearText(
77         message, duration));
78 }
79
80 /// <summary>
81 /// Coroutine to display text, wait, and then clear it.
82 /// </summary>
83 /// <param name="message">The text message to display.</param>
84 /// <param name="duration">How long to display the message in seconds
85 /// .</param>
86 private IEnumerator DisplayAndClearText(string message, float duration
87     )
88 {
89     // Set the text

```

```
81     _activeTextMesh.text = message;
82     _activeTextMesh.gameObject.SetActive(true); // Ensure the text
           GameObject is active
83
84     // Wait for the specified duration
85     yield return new WaitForSeconds(duration);
86
87     // Clear the text and reset color
88     _activeTextMesh.text = "";
89     _activeTextMesh.color = _initialTextColor; // Revert to the
           original color
90     currentDisplayCoroutine = null; // Mark that the coroutine has
           finished
91 }
92
93 /// <summary>
94 /// Immediately clears any temporary feedback text and stops its timer
           .
95 /// Resets text color to its initial state.
96 /// </summary>
97 public void ClearTextImmediately()
98 {
99     if (currentDisplayCoroutine != null)
100     {
101         StopCoroutine(currentDisplayCoroutine);
102         currentDisplayCoroutine = null;
103     }
104     if (_activeTextMesh != null)
105     {
106         _activeTextMesh.text = "";
107         _activeTextMesh.color = _initialTextColor; // Reset color
108     }
109 }
110
111 /// <summary>
112 /// Sets the text display to a persistent message.
113 /// This text will remain until explicitly changed by another call to
           this or a temporary display method.
114 /// Any ongoing temporary text display will be stopped.
115 /// </summary>
116 /// <param name="message">The text message to display persistently.</
           param>
117 /// <param name="color">Optional: The color of the text. If null, uses
           the text's initial color.</param>
118 public void SetPersistentText(string message, Color? color = null)
```

```

119 {
120     if (_activeTextMesh == null) return; // Exit if no active text
        component
121
122     // Stop any ongoing temporary display coroutine
123     if (currentDisplayCoroutine != null)
124     {
125         StopCoroutine(currentDisplayCoroutine);
126         currentDisplayCoroutine = null; // Clear the coroutine
            reference
127     }
128
129     // Set the text and apply color
130     _activeTextMesh.text = message;
131     _activeTextMesh.color = color ?? _initialTextColor; // Apply the
        specified color, or initial color
132     _activeTextMesh.gameObject.SetActive(true); // Ensure the text
        GameObject is active
133 }
134
135 /// <summary>
136 /// Clears any currently displayed text (temporary or persistent) and
        resets its color to initial.
137 /// This will also stop any ongoing temporary text display.
138 /// </summary>
139 public void ClearAllText()
140 {
141     ClearTextImmediately();
142 }
143 }

```

Listing A.6: TextDisplay script used to display messages on text feedback components

```

1
2 using MixedReality.Toolkit.UX;
3 using MixedReality.Toolkit.SpatialManipulation;
4
5 /// <summary>
6 /// A subclass of the MRTK Dialog component that disables automatic
        following
7 /// and button-based dismissal.
8 /// </summary>
9 public class StaticDialog : Dialog
10 {
11     protected override void Awake()
12     {

```



```
13         // Call the base Awake method to ensure all base functionality is
           initialized correctly.
14         base.Awake();
15
16         // Immediately find and disable the SolverHandler and Follow
           components to prevent the dialog from following the user.
17         StopFollowing();
18     }
19
20     /// <summary>
21     /// Overrides the base method to prevent the dialog from automatically
22     /// dismissing after a button is pressed.
23     /// </summary>
24     public override void Dismiss()
25     {
26         // Intentionally overriding this to do nothing.
27     }
28
29     /// <summary>
30     /// A public method to be called from a button or another script
31     /// to manually dismiss the dialog.
32     /// </summary>
33     public void ManuallyDismiss()
34     {
35         // Calls the base Dismiss method to safely hide the dialog and run
           all necessary cleanup.
36         base.Dismiss();
37     }
38
39     /// <summary>
40     /// Disables the SolverHandler and Follow components to prevent the
           dialog
41     /// from following the user's head.
42     /// </summary>
43     private void StopFollowing()
44     {
45         // Find the solver handler on this GameObject.
46         var solverHandler = GetComponent<SolverHandler>();
47         if (solverHandler != null)
48         {
49             solverHandler.enabled = false;
50         }
51
52         // Find the follow component on this GameObject.
53         var follow = GetComponent<Follow>();
```

```
54     if (follow != null)
55     {
56         follow.enabled = false;
57     }
58 }
59 }
```

Listing A.7: StaticDialog script used to override MRTK Dialog behavior