TECHNICAL UNIVERSITY OF CRETE

DIPLOMA THESIS

An Augmented Reality System for Real-time Decision-Making in Flood Emergencies

Author:
Ioannis SAFRANOGLOU

Committee:
Aikaterini MANIA
Antonios DELIGIANNAKIS
Nikos GIATRAKOS

A thesis submitted in fulfillment of the requirements for the degree of Electrical and Computer Engineering

October 17, 2024

TECHNICAL UNIVERSITY OF CRETE

Abstract

Electrical and Computer Engineering

An Augmented Reality System for Real-time Decision-Making in Flood Emergencies

by Ioannis SAFRANOGLOU

Locomotion interfaces for head-worn augmented reality (AR) devices are critical for rescuers moving on-site during disaster responses, such as flood emergency management. Incorporating live, real-time data—instead of static information—in the form of event forecasting over time without overloading the user's field of view is crucial for effective decision-making. This thesis presents an innovative head-worn AR flood management system designed to enhance situational awareness and decision-making for emergency responders on the go. The system dynamically visualizes present and predicted water levels, as well as water speed and direction at specific future time intervals, in real-time and regardless of location, thereby improving flood management while rescuers are moving. Additionally, points of interest are displayed on the horizon, with those most likely to require assistance indicated by color variations, allowing rescuers to navigate and respond more effectively to emergency scenarios. The system also features gaze-enabled interaction, enabling hands-free operation, which is essential in physically demanding environments. The prototype was tested by professional firefighters in Innsbruck and Dortmund under simulated flood conditions. Feedback from these trials highlighted the potential of providing comprehensive, real-time AR visualization to enhance disaster response strategies, with participants reporting improved situational awareness, faster decision-making, and increased operational efficiency. This research contributes to the field of disaster management by offering a practical solution that leverages advanced AR technology to meet the critical needs of emergency responders in real-time flood scenarios.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Aikaterini MANIA, for her invaluable guidance, support, and encouragement throughout the course of this research. Her expertise and insights have been instrumental in shaping this thesis.

My sincere thanks go to the entire SURREAL lab for their collaboration and assistance. Their camaraderie and dedication created an inspiring environment that greatly contributed to the success of this project.

I am also thankful for the opportunity provided by the CREXDATA project. Working on this project allowed me to develop my application and thesis simultaneously, offering a unique platform to contribute to innovative research in augmented reality and disaster management.

Finally, I wish to express my heartfelt appreciation to my family for their unwavering support and encouragement. Their love and confidence in me have been a constant source of motivation throughout my academic journey...

Contents

List of Figures

Chapter 1

Introduction

1.1 Brief Overview

Flood emergencies present numerous challenges for emergency responders, particularly when it comes to real-time decision-making and navigation through dynamic environments. Responders must rely on tools that provide accurate, up-to-the-minute information to maintain situational awareness and effectively manage their operations. However, traditional systems often fail to deliver real-time information or are limited by the need for handheld operation. Augmented Reality (AR), particularly head-worn AR, provides a promising solution by overlaying digital information directly into the user's field of view, allowing hands-free interaction and enhanced decision-making while on the move.

Locomotion affordances while deploying head-worn AR play a pivotal role in enhancing the effectiveness of AR navigation and wayfinding. Navigation interfaces for AR while moving are critical in disaster response scenarios (Xu et al., 2024), including flood emergency management (Barmpas Zachariadis, Nikolaos, 2020), as well as firefighting (Chalimas and Mania, 2023). Virtual Reality (VR) has proven effective for rescue training simulations (Haskins et al., 2020; Sermet and Demir, 2018) and remote flood rescue planning (Mol, Botzen, and Blasch, 2022), but AR is superior for on-site operations, facilitating real-time visualization of digital information superimposed onto physical objects without obstructing the user's view (Haynes, Hehl-Lange, and Lange, 2018; Smit, Voûte, and Verbree, 2021). AR's ability to improve situational awareness and operational efficiency has already been demonstrated in firefighting applications using thermal imaging and live tracking (Chalimas and Mania, 2023).

For flood emergencies, AR enhances spatial awareness and assists first responders by facilitating access to critical information in hazardous environments (Kourogi and Kurata, 2003; Nescher and Kunz, 2012). Previous research on mobile AR-based visualization of coastal erosion and sea level rise focused on static data in predefined locations (Katsiokalis, Ragia, and Mania, 2020; Sarri et al., 2022), while water visualization in mobile AR has been primarily implemented using handheld cameras, constraining rescuers' operations (Haynes, Hehl-Lange, and Lange, 2018; Erra et al., 2018; Andrade et al., 2022). Visualizing real-time data for spatial flood forecasting, responder locations, and points of interest (POIs) while on the move remains a challenge, as it is essential to provide crucial information in head-worn, hands-free AR without overloading the user's field of view (Daskalogrigorakis et al., 2022).

This thesis presents an innovative head-worn AR flood management system designed to support wayfinding by enhancing situational awareness and decision-making for emergency responders on the go. The system dynamically visualizes water levels, water speed, and water direction in real-time at any location, improving flood management. Environmental monitoring is refined, allowing responders to gain a clearer understanding of potential hazards. An integrated digital map offers a comprehensive view of the surroundings, enabling responders to anticipate flood behavior while moving at any location. The system prototype was tested by professional firefighters in Innsbruck and Dortmund, highlighting its potential for providing comprehensive, real-time AR visualization to enhance disaster response strategies.

FIGURE 1.1: Firefighter testing the AR system.

The specific contributions of this thesis include:

- An AR system combining a server and head-worn AR device, displaying predicted water levels, water velocity, and water direction in real-time at future time intervals, improving situational awareness and wayfinding for emergency responders.
- Integration of GPS data, from mobile platforms and Kafka topic, with head-worn AR to visualize critical POIs (such as schools and hospitals) at far distances and hazards such as manholes in close proximity. This enhances navigation, safety, and emergency planning.
- Uncertainty visualization for predicted flood levels over time, offering responders critical information about possible risk scenarios.
- Gaze interaction and gestures for hands-free operation, crucial in physically demanding environments.

- Proof-of-concept testing by professional firefighters in urban areas of Innsbruck and Dortmund, highlighting the system's potential to enhance navigation and wayfinding during flood emergencies.

Feedback from the trials with firefighters emphasized the system's capacity to improve operational efficiency, enhance situational awareness, and expedite decision-making in real-time. The system's ability to provide continuous updates about current and predicted water conditions makes it a valuable tool for flood emergency management, empowering responders to make better-informed decisions in dynamic and dangerous environments.

1.2 Thesis Structure

Chapter 1: Introduction

This chapter provides an overview of the thesis, including the problem statement, objectives, and motivation behind the development of an AR-based flood visualization system. It also includes a brief description of the project's context and goals.

Chapter 2: Background/Research Overview

This chapter introduces key concepts, including protocols and services, which plays a role in the system's backend. It also provides an overview of Augmented Reality (AR), covering its history, use cases, and human-computer interaction aspects, and specific research in the context of flood management and emergency response.

Chapter 3: Technological Background and Definitions

This section delves deeper into the tools and technologies used in the development of the AR application. It covers game engines (especially Unity), the Mixed Reality Toolkit (MRTK3), and the Microsoft Hololens 2, alongside relevant SDKs such as Mapbox for geolocation services.

Chapter 4: Server and Data Communication

This chapter explains the system backend architecture and communication pipelines, focusing on how the Hololens 2 and mobile devices connect to the server. It outlines the flow of GPS data, WebSocket communication, and Kafka integration, while also addressing synchronization and reconnection mechanisms for ensuring continuous data flow.

Chapter 5: Implementation

The most detailed chapter, this section covers the implementation of the AR system. It includes the system AR architecture, user interface, and various components like POIs, water visualization, map integration, and interaction methods. This chapter provides a breakdown of how each feature was developed and its technical challenges, such as the calibration process, prediction sliders, and POI spawning.

Chapter 6: Evaluation

This chapter presents the trials conducted in Innsbruck and Dortmund, providing feedback and observations from the firefighting experts who tested the system. It includes visual data illustrating the system's usability, along with challenges identified during the trials, such as hardware limitations and feedback on specific AR functionalities.

Chapter 7: Conclusion and Future Work

The final chapter summarizes the findings from the development and testing phases of the project. It reflects on the AR system's contributions to enhancing situational awareness for emergency responders, and outlines potential future improvements, such as better system optimization, enhanced POI functionalities, and resilience in communication-limited scenarios.

Bibliography

This section includes references to all the literature and sources cited throughout the thesis.

Chapter 2

Backround/Research Overview

2.1 Introduction

This chapter provides the necessary background and research context for understanding the development and implementation of the Augmented Reality (AR) system for real-time decision-making in flood emergencies. The system relies on a combination of real-time data streaming, predictive analytics, and the use of advanced AR hardware and software platforms. To fully grasp the functionality of the system, it is essential to explore the core technologies and protocols that enable its operation.

One key component is the integration of GPS data and real-time environmental information, which is sent from a custom mobile application to the system's server via WebSocket. The WebSocket protocol ensures reliable and low-latency communication, allowing the system to transmit accurate, up-to-the-minute location data during emergency operations. Additionally, the system is connected to a Kafka topic using the KafkaJS library, which facilitates the processing and handling of large-scale, real-time data streams. This setup allows the AR system to incorporate live flood data and predictive analytics into the visualization displayed on the Hololens 2.

On the Unity side, the WebSocket-Sharp library is used to manage data transmission between the server and the AR application running on the Hololens 2, ensuring that emergency responders can interact with real-time data seamlessly. The combination of these technologies—WebSocket for communication, Kafka for data streaming, and Unity for AR visualization—ensures that the system provides comprehensive, real-time situational awareness for first responders during flood emergencies.

In this chapter, we will outline the key protocols, services, software platforms, and hardware used in the development of this AR system. We will also review relevant research and existing systems in AR and disaster management to provide context for how this work builds upon and extends prior advancements. By understanding these technological foundations, we can appreciate how this AR system enhances decision-making and operational efficiency in dynamic and hazardous environments.

2.2 Protocols and Services

The system's ability to deliver real-time situational awareness during flood emergencies relies on several core protocols and technologies. This section covers the primary services and communication protocols that enable the integration of live

GPS data, flood information, and real-time visualization in the AR interface. These technologies include WebSocket for communication, a custom mobile application for GPS tracking, Unity for AR visualization, and Kafka for large-scale data streaming.

2.2.1 Global Positioning System (GPS)

Global Positioning System (GPS) is a satellite-based navigation system that allows users to determine their precise location anywhere on Earth. It operates through a constellation of at least 24 satellites that orbit the Earth, constantly transmitting signals that GPS receivers on the ground can pick up. By calculating the time it takes for these signals to travel from multiple satellites, a GPS receiver can triangulate its exact position, typically within a few meters of accuracy.

The development of GPS began in the 1970s by the U.S. Department of Defense, primarily for military applications. The system became fully operational in 1993 and has since been opened for civilian use. Initially, GPS was designed to provide precise navigation and positioning data for military assets, including ships, aircraft, and ground forces. However, in the early 2000s, GPS became widely available to the public after the U.S. government disabled "Selective Availability," which previously limited the accuracy of civilian GPS. Over time, the accuracy and availability of GPS have improved significantly, thanks to advancements in satellite technology and receiver design.

Typical Uses of GPS

GPS is now an integral part of modern society and has a wide array of applications. It is commonly used in:

- **Navigation:** GPS is a cornerstone technology in personal and commercial navigation systems, helping people travel from one place to another efficiently.
- **Geolocation and Mapping:** GPS is essential in mapping services like Google Maps, providing real-time location data and route planning.

- **Surveying and Geophysics:** GPS is also used in land surveying, geophysical measurements, and scientific research to determine precise geographic coordinates.
- **Emergency Response:** GPS plays a critical role in emergency services, where it helps track the location of emergency vehicles and personnel during operations.

In this project, GPS is used to track the location of emergency responders during flood emergencies. Since the Hololens 2 does not have built-in GPS functionality, a custom mobile application was developed to bridge this gap. The mobile app continuously collects GPS data from the user's device, providing real-time updates on the user's location.

The mobile application connects to the system's server through WebSocket and transmits GPS data at regular intervals. This data is processed on the server and sent to the Hololens 2 AR system, where it is integrated into the AR environment. This ensures that emergency responders are constantly aware of their own location, as well as the locations of other team members. The visualization of real-time location data allows for more efficient navigation and coordination during rescue operations, making it easier for responders to assess the situation and take necessary actions quickly.

The GPS data is particularly valuable in flood scenarios where the environment is rapidly changing. As the floodwaters rise, responders need up-to-the-minute location data to navigate safely and avoid hazardous areas. By providing this data in real time, the system enhances situational awareness, reduces the risk of responders entering dangerous zones, and helps coordinate their movements more effectively. The GPS data is also integrated with flood prediction models, allowing users to visualize how water levels will change in relation to their location.

2.2.2 WebSocket Protocol

WebSocket is a communication protocol that provides full-duplex communication channels over a single, long-lived connection between a client and a server. Unlike traditional HTTP connections, which are request-response-based and require new connections to be established for each interaction, WebSocket allows continuous, real-time communication between two parties without the overhead of repeatedly opening and closing connections.

Once a WebSocket connection is established, data can flow freely in both directions—server to client and client to server—without the need for additional handshakes. This makes WebSocket an ideal protocol for applications that require frequent updates or real-time interactions, such as online gaming, live chat, stock market tickers, and IoT applications.

The WebSocket protocol was standardized in 2011 as RFC 6455 by the Internet Engineering Task Force (IETF). It was developed as a solution to the limitations of HTTP, especially in scenarios requiring real-time communication. Prior to WebSocket, developers relied on techniques like long polling and Comet to simulate real-time interaction, but these approaches were inefficient and consumed excessive network resources.

WebSocket emerged as a much more efficient solution, offering reduced latency and bandwidth usage, as well as simplifying the development process for real-time web applications. It has since become a standard tool in modern web development, supported by all major browsers and many server-side technologies.

Typical Uses of WebSocket

WebSocket is widely used in applications where real-time data exchange is crucial, such as:

- **Real-time Web Applications:** WebSocket powers real-time interactions in applications like live sports updates, online multiplayer games, and collaborative tools.
- **Financial Services:** Stock trading platforms and cryptocurrency exchanges use WebSocket to provide users with real-time price updates and trade notifications.
- **Internet of Things (IoT):** In IoT ecosystems, WebSocket enables real-time data streaming from sensors to cloud servers or mobile devices.
- **Instant Messaging and Live Chats:** WebSocket is the backbone of modern instant messaging platforms, where it supports real-time message delivery and notifications.

In this AR flood management system, WebSocket serves as the communication layer between the custom mobile application, the server, and the Unity-based AR interface on the Hololens 2. It allows for continuous real-time data exchange, ensuring that GPS data, water level information, and other critical environmental metrics are transmitted efficiently.

By maintaining a persistent connection, WebSocket ensures that the AR system receives real-time updates without the need to repeatedly initiate new connections. This is particularly important in flood emergency scenarios, where low-latency and high-frequency data updates are critical for making timely decisions. The protocol's ability to handle multiple connections simultaneously allows the server to manage data from several sources and transmit it to the Hololens 2 for real-time visualization.

2.2.3 Kafka

Apache Kafka is a powerful distributed event streaming platform, widely used for real-time data processing and pipeline management. Initially developed by LinkedIn in 2011, Kafka was created to address the challenge of processing vast amounts of event data in real-time. Over time, Kafka has evolved into a robust platform capable of handling millions of messages per second, making it an essential tool for any application that requires continuous data flow and rapid decision-making capabilities.

Kafka works by allowing different systems to publish and subscribe to streams of records, known as topics. These topics are partitioned to enable scalability across multiple servers, ensuring that even massive volumes of data can be processed efficiently. This partitioning also allows Kafka to be fault-tolerant, making it ideal for critical real-time applications where data integrity and performance are crucial.

A key reason Kafka stands out is its ability to store event streams durably, allowing data consumers to access the data at any time. This flexibility, combined with its low-latency capabilities, has led Kafka to become a backbone in modern data architectures. Many industries, from finance to telecommunications, rely on Kafka for tasks such as real-time analytics, monitoring, and data integration.

In this AR flood management system, Kafka plays a pivotal role in managing large-scale data streams related to flood conditions. Through its distributed nature, Kafka enables the system to ingest and process continuous data feeds from various sources, including water level sensors and predictive flood models. By using the KafkaJS library, the server is able to connect to Kafka topics, retrieve real-time data, and relay that data to the Hololens 2 interface via WebSocket. This allows emergency responders to receive live updates on water levels, flow rates, and other environmental conditions, which are critical for making informed decisions in real-time.

Kafka's ability to handle high-throughput data streams ensures the system can operate smoothly even during large-scale flood events. By leveraging Kafka's inherent scalability, the system can seamlessly handle the growing volume of data as more sensors or data sources are added. This is essential in emergency situations, where time-sensitive data must be processed and delivered without delays. The integration of Kafka into the AR system ensures that emergency responders are not only informed of the current situation but also receive predictive analytics that help them anticipate changes in flood conditions.

In summary, Kafka's role in this system goes beyond simple data transmission—it acts as the core infrastructure that supports real-time situational awareness by providing reliable and scalable data streaming. By ensuring that the right data is available at the right time, Kafka enables the AR system to offer a fully informed view of the evolving flood landscape, helping first responders to act quickly and safely.

2.2.4 Node.js

Node.js is a powerful open-source, cross-platform JavaScript runtime environment built on Chrome's V8 engine. It was designed to execute JavaScript code outside of a browser, primarily for building fast and scalable server-side applications. The hallmark of Node.js is its event-driven, non-blocking I/O model, which allows developers to handle large numbers of simultaneous connections efficiently. This makes Node.js a popular choice for building real-time, data-intensive applications that must handle significant network traffic, such as web servers, API services, and streaming applications.

History and Development of Node.js Node.js was first released in 2009 by Ryan Dahl, and its development was motivated by the need for a server-side environment that could efficiently handle asynchronous operations. At the time, traditional server models—such as Apache—handled each client request using a new thread,

which was often resource-heavy and not ideal for real-time applications with high concurrency demands. Node.js solved this problem by using a single-threaded event loop, which can handle thousands of concurrent connections without the overhead of creating multiple threads for each request.

Since its initial release, Node.js has seen widespread adoption across industries. Its popularity grew further due to its use of JavaScript, which allows developers to use the same language for both client-side and server-side programming. Node.js has evolved to support a rich ecosystem of libraries and tools, thanks to its package manager npm (Node Package Manager), which contains over a million open-source packages that developers can use to extend their applications.

Typical Uses of Node.js

Node.js is widely used in applications that require real-time communication and high performance:

- **Web Servers:** Node.js is often used to build web servers that serve API requests, dynamic web pages, or static assets. Its asynchronous I/O operations make it ideal for handling large numbers of requests simultaneously.
- **Real-Time Applications:** Node.js shines in real-time applications, such as chat apps, online gaming, and collaboration tools, where rapid communication between server and clients is essential.
- **Streaming Services:** Due to its event-driven nature, Node.js is a great fit for applications that involve data streaming, such as video and audio streaming platforms.
- **Microservices and APIs:** Node.js is a popular choice for building microservices and API-based architectures due to its lightweight and efficient nature. It integrates well with other technologies and is frequently used for REST and GraphQL APIs.

Node.js is used in the AR flood management system as the core server-side platform. Its ability to handle multiple simultaneous connections efficiently makes it ideal for real-time data transmission between the mobile app, the AR interface, and other services. The non-blocking I/O model ensures that the system can process and relay real-time data, such as GPS updates and flood information, without any delays or performance bottlenecks.

The choice of Node.js also allows for flexibility and scalability, enabling the system to handle growing data sources and user connections as the project expands. The details of how Node.js interacts with other components like WebSocket and external

data streams will be explored in a later chapter dedicated to server architecture and communication.

2.2.5 Conclusion of Protocols and Services

In this section, we explored the core protocols and services that form the backbone of the AR flood management system. These technologies—GPS for location tracking, WebSocket for real-time communication, Kafka for large-scale data streaming, and Node.js as the server environment—are integral to ensuring the system operates efficiently in real-time. By leveraging these well-established protocols and technologies, the system is able to provide emergency responders with timely and accurate information during flood emergencies. In the following sections, we will continue to build on this foundational understanding by exploring the role of Augmented Reality and its related technologies in creating an intuitive and immersive user experience.

2.3 Augmented Reality

Augmented Reality (AR) is a technology that overlays digital information onto the physical world, enabling users to interact with both real-world and virtual objects in real-time. Unlike Virtual Reality (VR), which immerses users in a completely digital environment, AR enhances the user's existing environment by adding contextual digital content. This digital content can be anything from simple 2D overlays, such as text and images, to complex 3D objects and animations that interact with real-world elements. AR can be viewed as a continuum, with real-world environments at one end and completely virtual environments at the opposite end.

Different types of reality

- **Real Environment (RE):** It is the environment in which we live and is governed by the laws of physics.
- **Augmented Reality (AR):** Physical reality in which participants also see virtual elements.
- **Mixed Reality (MR):** It is a virtual reality in which the participants also see real elements.
- **Virtual Reality (VR):** Represents a synthetic world in which the participant is completely immersed.

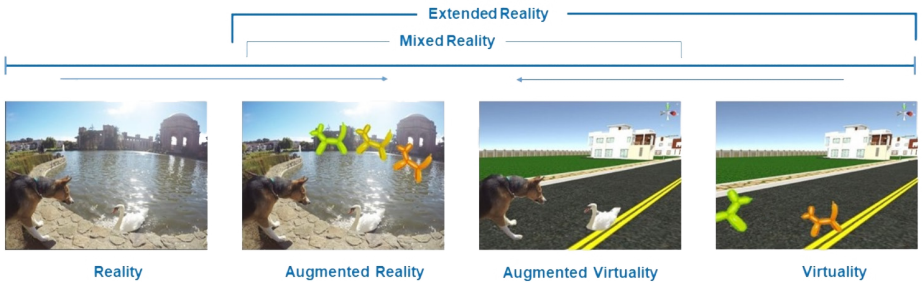


FIGURE 2.1: Types of Reality

AR systems can be deployed on various devices, including smartphones, tablets, and head-mounted displays (HMDs) such as Microsoft Hololens and Google Glass. While early AR applications were limited by the capabilities of mobile devices, the advent of more sophisticated hardware has significantly expanded the potential uses of AR in a wide range of industries.

Key Characteristics of AR AR operates based on three core characteristics:

- **Combining Real and Virtual Worlds:** AR allows users to see the physical world with digital elements superimposed onto it. This is achieved through a combination of sensors, cameras, and display technologies that track the user's movements and position in space, ensuring that virtual content aligns correctly with the physical environment.
- **Interactive in Real Time:** One of the defining features of AR is its real-time interaction. The user can engage with virtual objects, which respond to inputs such as gestures, voice commands, or gaze direction, making the experience dynamic and engaging.
- **Three-Dimensional Registration:** AR content must be spatially aware to function correctly. This means the digital objects need to be anchored to specific points in the real world and should remain in place even as the user moves around. This capability is especially important for applications in fields like healthcare, engineering, and disaster management, where precision and context are key.

AR technology bridges the gap between the digital and physical worlds, enhancing users' interactions with their surroundings by providing context-specific information or interactive elements. The flexibility of AR makes it useful across various domains, from entertainment to industrial applications.

Applications of AR AR has seen a rapid expansion of use cases across several fields, including:

- **Entertainment and Gaming:** AR has gained widespread popularity in gaming with the advent of applications like Pokémon GO and AR-based board games, which superimpose game elements onto the real world, blending physical and digital experiences.
- **Education:** In the classroom, AR offers interactive learning experiences, allowing students to explore complex subjects like anatomy or physics in an immersive and hands-on manner.
- **Healthcare:** AR is transforming the healthcare industry by assisting surgeons with real-time overlays of medical images during operations. AR can also be used to train medical professionals through simulations that allow them to visualize anatomical structures in three dimensions.
- **Manufacturing and Maintenance:** AR is increasingly used in industrial settings to overlay real-time diagnostics, instructions, and schematics onto machinery and equipment, making maintenance and troubleshooting faster and more efficient.

(A) Education (B) Healthcare (C) Manufacturing

FIGURE 2.2: Applications of AR

2.3.1 Brief History of AR

The concept of Augmented Reality (AR) has its roots in the 1960s, when early pioneers of computer graphics began exploring ways to overlay digital images onto the physical world. Over the years, AR has evolved from basic experimental systems into a powerful technology with wide-ranging applications across industries. This section provides an overview of the key milestones in the development of AR, from its early beginnings to modern-day use cases.

Early Beginnings

AR's history can be traced back to 1968, when Ivan Sutherland, often referred to as the "father of computer graphics," developed the first head-mounted display (HMD) system, known as the Sword of Damocles. Although primitive by today's standards, Sutherland's work laid the groundwork for future advancements in both AR and Virtual Reality (VR). The device was capable of rendering simple wireframe graphics that appeared to be superimposed over the user's real-world environment, marking the first step toward merging digital and physical worlds.

Development through the 1990s

The term "Augmented Reality" was coined in 1990 by Tom Caudell, a researcher at Boeing, while developing AR systems to assist factory workers in assembling aircraft. The AR system Caudell worked on displayed digital schematics and instructions over physical aircraft components, allowing workers to see real-time visual guides while working on the factory floor.

In the 1990s, AR saw further development in military and industrial applications. One notable example is the U.S. Air Force, which used AR to provide aircraft mechanics with digital overlays of wiring diagrams and other critical information, helping them maintain and repair aircraft more efficiently. Around the same time, research into AR's potential in other professional domains, such as medicine and architecture, began gaining traction.

Mainstream Adoption and Mobile AR

In the early 2000s, AR began to make its way into the consumer market, primarily through mobile AR applications. With the proliferation of smartphones equipped with cameras and GPS, developers could create AR experiences that used the phone's display to superimpose digital content onto the physical world. One of the earliest

AR tools, ARToolKit, enabled developers to create marker-based AR applications, which could track specific visual markers and overlay digital images in real time.

The introduction of Layar in 2009 marked a significant step in mobile AR, as it allowed users to view real-world environments through their smartphone's camera while receiving contextual information overlaid on the display. Around this time, the gaming industry also began exploring AR with titles like Ingress, an AR-based game that utilized real-world locations as part of the gameplay.

AR's Popularization with Pokémon GO

The global success of Pokémon GO in 2016 marked a turning point for AR's popularity. Developed by Niantic, the game allowed players to catch virtual Pokémon characters overlaid on real-world environments using their smartphone cameras. Pokémon GO introduced AR to millions of users worldwide, demonstrating AR's potential for engaging and interactive experiences.

FIGURE 2.3: Pokémon GO

Modern AR Devices and Applications

While early AR systems were primarily mobile-based, the development of head-worn AR devices significantly expanded the technology's potential. The release of Google Glass in 2013 and the Microsoft Hololens in 2016 paved the way for AR applications in professional settings. These devices allowed users to interact with digital content hands-free, making AR a valuable tool in fields like healthcare, architecture, education, and disaster management.

The Microsoft Hololens 2, in particular, has become one of the most advanced AR platforms available, offering users a fully immersive AR experience. Equipped with spatial mapping, gaze tracking, and hand gesture recognition, the Hololens 2 is used in a variety of industries, from healthcare (where it aids in surgical planning) to emergency response (where it provides real-time data visualizations in critical situations).

AR in Today's World

Today, AR continues to evolve and expand its presence across industries. From its humble beginnings in military and industrial applications to mainstream consumer use, AR has become a versatile tool with countless applications. Modern AR systems are now being integrated with Artificial Intelligence (AI) and Machine Learning (ML), further enhancing their ability to provide context-aware and intelligent overlays.

In particular, AR's role in disaster management and emergency response has shown significant promise. By providing real-time data overlays, such as flood levels and hazard zones, AR systems can improve situational awareness for emergency responders, helping them make informed decisions during crises.

2.3.2 Use Cases of Augmented Reality (AR)

Augmented Reality (AR) has grown significantly in scope and application since its inception, finding use in a wide array of industries and domains. By overlaying digital information onto the real world, AR enhances user experiences, providing contextual information and interactive elements that make both consumer and professional tasks more efficient. This section outlines some of the key use cases of AR across industries and highlights its unique potential in fields such as emergency response and disaster management.

Entertainment and Gaming

One of the most well-known uses of AR is in the entertainment and gaming industries. AR allows players to interact with virtual characters and objects within the real world, creating immersive and engaging experiences. A notable example of AR in gaming is Pokémon GO (2016), which allowed players to explore their physical environment while capturing virtual creatures superimposed on real-world locations. This game popularized AR on a global scale, demonstrating the technology's potential to merge digital and physical worlds for enhanced player experiences.

Beyond gaming, AR is also being used in theme parks, interactive movies, and live events to create more engaging and immersive entertainment experiences. As AR technology continues to evolve, its applications in the entertainment industry are expected to grow, with new opportunities for interactive storytelling and personalized content.

Education

In the field of education, AR is being used to transform the traditional learning experience by creating interactive and immersive learning environments. Students can explore complex subjects, such as anatomy, physics, or history, through AR-based visualizations that allow them to interact with 3D models, simulations, and virtual environments.

For example, in science education, students can use AR to conduct virtual experiments, visualize molecular structures, or explore the human body in three dimensions. AR also plays a key role in historical education, where students can use their devices to view recreations of ancient landmarks or historical events within their physical environment, making learning more engaging and memorable.

Healthcare

In healthcare, AR is revolutionizing how medical professionals approach diagnostics, training, and surgical procedures. Surgeons can now use AR to overlay MRI or CT scan data directly onto a patient during surgery, allowing for real-time guidance and improved accuracy. AR is also being used in medical training, where students and professionals can visualize and interact with 3D anatomical models in augmented environments, enhancing their understanding of complex medical concepts.

The application of AR in rehabilitation is another emerging field. Patients undergoing physical therapy can use AR systems to guide them through exercises, with real-time feedback on their movements. AR helps improve patient engagement and outcomes by providing visual cues and progress tracking.

Disaster Management and Emergency Response

One of the most promising applications of AR is in disaster management and emergency response. In these scenarios, AR can provide first responders with critical, real-time information overlaid onto their physical environment. For example, during a flood emergency, AR systems can display current water levels, flood predictions, and hazard zones directly in the responder's field of view, improving situational awareness and enabling more informed decision-making.

AR systems can also be used to display evacuation routes, mark points of interest (e.g., people in need of assistance), and provide navigation in complex or hazardous environments. This capability is especially valuable in dynamic situations where conditions change rapidly, as AR allows responders to access up-to-date information without needing to look away from their surroundings.

The ability to visualize real-time data in the field—whether through head-mounted displays like the Microsoft Hololens 2 or through mobile AR apps—can enhance both the safety and effectiveness of emergency response efforts. By integrating live sensor data, such as weather conditions or GPS tracking, AR systems can provide responders with a comprehensive understanding of the evolving situation, helping them prioritize tasks and allocate resources more efficiently.

2.3.3 Human-Computer Interaction (HCI) in AR

Human-Computer Interaction (HCI) in Augmented Reality (AR) involves the design and study of how users engage with AR systems to interact with digital content superimposed onto the physical world. Effective HCI is vital to ensuring that AR systems are intuitive, responsive, and seamless, especially when used in demanding environments such as emergency response, where users must process critical information without distractions or delays.

HCI in AR has evolved alongside the development of AR hardware and software, focusing on how to best integrate digital content into a user's physical surroundings while maintaining natural and efficient interactions. This is particularly important in head-mounted AR systems, where the user's focus remains in the real world while digital elements are overlaid in real-time.

Interaction Methods in AR

In AR systems, several interaction methods are used to allow users to engage with virtual objects in the real world. The three primary methods of interaction in head-worn AR systems are gesture recognition, voice commands, and gaze tracking.

- **Gesture Recognition:** Gesture-based interaction allows users to manipulate virtual objects using their hands or body movements. In head-mounted displays (HMDs) like the Microsoft Hololens 2, users can reach out and interact with digital objects by performing gestures such as pinching, tapping, or dragging. These gestures are tracked using cameras and sensors embedded in the device, allowing for a highly intuitive and hands-free interaction experience. For example, users can "grab" a digital tool or element and move it around in space as if it were a physical object.

  Gesture-based interactions are especially useful in emergency response scenarios, where responders need to keep their hands free for other tasks while accessing critical information. The ability to manipulate AR content through simple hand gestures allows users to interact with the system without needing to handle physical controllers or devices, improving both usability and efficiency.

- **Voice Commands:** Voice commands are another key method of interaction in AR systems, particularly in environments where hands-free operation is essential. AR devices like the Hololens 2 incorporate speech recognition technology that enables users to issue commands by simply speaking. This is especially useful in scenarios where the user's hands are occupied or where gesture-based input may be impractical due to environmental conditions.

  For instance, in a flood emergency, a responder could use voice commands to switch between different data views (e.g., flood levels, hazard zones) or call up specific pieces of information, such as the location of a nearby evacuation route, without interrupting their physical tasks. Voice commands provide a convenient and efficient way to control AR systems in fast-paced, dynamic environments.

- **Gaze Tracking:** Gaze tracking is a highly intuitive method of interaction that uses the user's eye movements to control the system. In gaze-based interfaces, the AR device tracks where the user is looking and can trigger interactions based on the focus of the user's gaze. For example, the Hololens 2 uses eye-tracking sensors to allow users to select objects by simply looking at them, reducing the need for more complex hand or voice interactions.

  This interaction method is particularly effective in applications that require rapid focus and minimal physical input. In disaster management situations, gaze tracking allows users to quickly access and interact with critical data while maintaining their situational awareness. Gaze tracking also minimizes cognitive load by enabling hands-free interaction with AR content, making it ideal for real-time decision-making environments.

Importance of Intuitive Interfaces in Critical Environments

In environments such as disaster management and emergency response, where decisions need to be made quickly and accurately, the design of AR interfaces must

prioritize usability and efficiency. The goal is to provide users with a seamless, immersive experience that allows them to interact with digital content naturally, without causing distraction or cognitive overload.

In such critical situations, poorly designed interfaces can be detrimental, as responders may miss important information or struggle to interact with the system while navigating complex environments. An intuitive AR interface should:

- **Minimize distractions** by keeping the digital content clean, concise, and contextually relevant.
- **Enhance situational awareness** by providing real-time data that is easily accessible and actionable.
- **Support natural interactions** through gestures, voice commands, and gaze tracking, allowing users to focus on their surroundings while interacting with the system.

Microsoft Hololens 2 is one of the leading devices in this regard, as it offers a range of interaction options designed to support real-time decision-making in dynamic environments. By incorporating gaze, gesture, and voice-based interaction methods, the Hololens 2 enables users to interact with complex datasets and virtual objects while keeping their attention on their physical surroundings. This makes it especially valuable for applications such as flood management, where responders need to visualize critical information in real-time while staying engaged with their environment.

2.4 Augmented Reality in Flood Management

Augmented Reality (AR) has been increasingly adopted in emergency response scenarios to enhance situational awareness and improve operational efficiency (Sebillo et al., 2015; Sebillo et al., 2016; Nunes et al., 2019; Siu and Herskovic, 2013). AR systems allow first responders to access critical information in real-time, directly overlaid onto their physical environment. By presenting digital data within the responder's field of view, AR reduces the need to consult handheld devices or paper maps, enabling hands-free operation and improving focus on the task at hand.

Traditionally, mobile AR systems have been developed to assist responders during emergencies, offering a range of features. However, mobile AR systems occupy the user's hands, limiting their ability to perform critical tasks while interacting with the device. For instance, Campos et al. designed an AR mobile system that assisted first responders during emergency evacuations by superimposing location data and navigational instructions over a mobile interface (Campos et al., 2019). While effective in certain contexts, these systems have limitations when users need to remain hands-free to carry out physical tasks.

To overcome these limitations, head-worn AR systems have emerged as a powerful tool for emergency response, offering gaze-driven interaction and hands-free operation (Sainidis et al., 2021; Wani, 2013). These systems allow first responders to visualize critical information directly within their environment. For example, Nelson et al. developed a head-worn AR triage tool designed to assist responders during mass casualty incidents (Rae Nelson et al., 2022; Nelson et al., 2022) by presenting triage information directly in the responder's field of view. It focused exclusively

on triage scenarios and did not address other in-the-field tasks. In contrast, this system deploys head-worn AR for hands-free, gaze and gesture driven interaction by firefighters, visualizing real-time data including present and predicted water levels, including prediction uncertainty, enhancing situational awareness and operational efficiency for first responders.

2.4.1 Flood Visualization

Flood visualization tools have become essential in enabling emergency responders to manage and mitigate the impact of flooding (Haynes, Hehl-Lange, and Lange, 2018; Erra et al., 2018). Many of these systems have traditionally been designed for mobile devices, which can limit their effectiveness in emergency scenarios where quick, hands-free interaction is necessary. To address this limitation, head-worn AR systems have also been developed to provide more seamless, real-time flood visualization.

For instance, Sarri et al. designed an AR system that visualizes future sea level rise at coastal locations, aimed at enhancing public awareness of climate change (Sarri et al., 2022). However, their system relied on static data and pre-defined locations, and did not support user movement. The water visualization was stationary, meaning that if users moved away from the designated locations, the visualization did not update to reflect their new positions.

In contrast, our system dynamically adjusts the scanned mesh of the environment to follow the user, allowing for continuous scanning and visualization of water levels as the user moves. This enhances the user's ability to stay updated on real-time flood conditions while navigating different areas. Similarly, Wang et al. developed a head-worn AR system that created an interactive 3D model of urban flood scenarios. While this system did not provide on-location visualization, it allowed users to manipulate and explore flood data to support better decision-making (Wang et al., 2020).

Our AR-based flood visualization system goes further by projecting flood data directly into the user's real-world environment, visualizing flood levels at specific locations around the user to indicate the potential impact of flooding. This capability is crucial for first responders, who need to maintain situational awareness in dynamic and hazardous conditions. Rydvanskiy et al. also developed a head-worn AR flood visualization system, which evaluated the system's usability in flood risk management (Rydvanskiy and Hedley, 2021). Their approach integrated 3D geo-spatial data into the physical space of the user, using a map to visualize flood scenarios.

Our system builds upon these advancements by combining AR with a robust server infrastructure, providing responders with real-time flood data in a hands-free, immersive format. By effectively visualizing water levels, flow velocity, and direction over time, our system supports first responders in the field, allowing them to receive critical information while remaining mobile. This dynamic visualization ensures that responders have access to the most up-to-date data, enhancing their ability to respond quickly and efficiently to flood emergencies.

2.4.2 Uncertainty Visualization

Uncertainty visualization is a critical component of data representation that helps users better understand the reliability and variability of the information presented.

In emergency scenarios, it is essential that responders comprehend the uncertainty inherent in the data, as it directly impacts their decision-making processes. Properly visualizing uncertainty allows for a clearer understanding of data variability and reliability, which is crucial for making informed decisions in dynamic, high-risk environments (Potter, Rosen, and Johnson, 2012).

When managing floods, uncertainty visualization techniques such as heatmaps and ensemble visualizations are often employed. For instance, probabilistic flood maps typically use heatmaps to represent flood risk based on levels of rainfall. These heatmaps might vary in color from blue to red, indicating increasing levels of risk (Gomez et al., 2024). Ensemble visualizations, on the other hand, utilize multiple data sources and color coding to display the likelihood of various outcomes, providing users with a more comprehensive view of possible flood scenarios (MacEachren et al., 2012).

By using these techniques, responders can visualize not only the most likely outcomes but also the full range of potential risks and scenarios. This enables a more holistic approach to risk management, allowing users to anticipate different possible outcomes and adjust their strategies accordingly. Deploying uncertainty visualization in AR, particularly in systems designed for rescue operations, offers significant advantages in terms of real-time decision-making and planning. By integrating these visualizations with real-time data streaming, AR systems provide emergency responders with a clearer understanding of risk levels and future outcomes, helping them to allocate resources more effectively (Oliveira et al., 2015).

In our AR system, we enhance situational awareness by displaying the prediction certainty of the flood visualization. This is achieved through the use of a slider interface, which allows users to select specific time intervals and visualize both the present and future water levels. The system also indicates the degree of uncertainty by utilizing a numeric probability value that changes based on the predicted reliability of the data. Additionally, Points of Interest (POIs) are color-coded to reflect risk levels, allowing responders to quickly assess the relative danger of various locations and prioritize their response efforts accordingly.

Chapter 3

Technological Background and Definitions

3.1 Game Engines

A game engine is a comprehensive software framework used to create real-time 2D and 3D applications, including video games, simulations, and increasingly, immersive Augmented Reality (AR) and Virtual Reality (VR) experiences. Game engines offer developers a set of essential tools for building, rendering, and managing complex virtual environments. These tools include graphical rendering systems, physics engines, scripting APIs, animation tools, and more, all of which are designed to streamline the development process and reduce the amount of repetitive coding required to build interactive applications.

At their core, game engines typically consist of several key components:

- **Rendering Engine:** The rendering engine is responsible for drawing the 2D and 3D visuals on the screen in real-time. It processes graphical assets and brings them to life by managing lighting, textures, and shading. In AR applications, the rendering engine ensures that virtual objects are seamlessly overlaid on the real-world environment.
- **Physics Engine:** This component simulates the behavior of objects according to the laws of physics. It handles gravity, collision detection, and other interactions that give virtual objects realistic behavior within the environment.
- **Scripting System:** Game engines offer scripting languages that allow developers to define how objects and characters behave within the environment. In most modern engines, scripting is done using accessible programming languages like C# (in Unity) or Blueprint Visual Scripting (in Unreal Engine), allowing for more rapid development.
- **Animation System:** Animation systems allow for the creation of realistic movement for characters and objects. This is particularly important in AR applications where interaction with virtual elements must feel smooth and natural to maintain immersion.
- **Input Management:** Game engines manage user inputs, whether from keyboard, mouse, touchscreens, or, in the case of AR, hand gestures, voice commands, and gaze tracking. These inputs are then mapped to actions within the application.

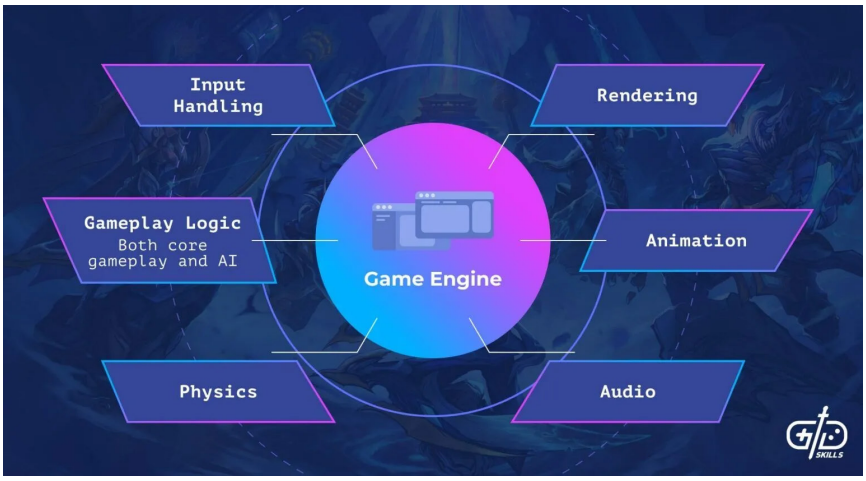


FIGURE 3.1: Components of a Game Engine

Modern game engines have become the backbone of many non-gaming applications, expanding into industries such as education, healthcare, and industrial simulation. For AR applications specifically, game engines need to support advanced features such as spatial mapping and device tracking, which allow virtual objects to be anchored in real-world spaces, making them an integral part of the physical environment.

Examples of Popular Game Engines for AR Development

- **Unity:** One of the most widely used game engines, Unity is favored for its cross-platform capabilities and its extensive library of assets and tools. Unity supports a wide range of platforms, including mobile devices, desktops, consoles, and head-worn AR devices like the Microsoft Hololens. Its accessible scripting environment (using C#) and robust AR Foundation make it a preferred engine for developing AR applications. Unity's flexibility allows developers to work on AR projects for different platforms without starting from scratch for each one.
- **Unreal Engine:** Known for its high-fidelity graphics, Unreal Engine is often used in projects that require visually intensive applications, such as AAA games and architectural simulations. Unreal's Blueprint Visual Scripting system allows for quick prototyping, but it may not be as well-suited for AR development as Unity due to fewer built-in tools for AR platforms, particularly on mobile devices. However, it still offers powerful tools for high-end AR development, especially in applications where visual realism is crucial.

Both Unity and Unreal Engine support core AR development needs, such as device tracking, spatial mapping, and plane detection, which are essential for creating immersive and interactive AR experiences. However, Unity's extensive plugin support (e.g., ARCore for Android, ARKit for iOS, and Windows XR for Hololens) and its large community make it more accessible for AR developers, especially those working on cross-platform projects.

	Unity	Unreal
Graphics	Physically-Based Rendering, Global Illumination, Volumetric lights after a plugin installed, Post Processing	Physically-Based Rendering, Global Illumination, Volumetric lights out of the box, Post Processing, Material Editor
Unique Features	Rich 2D support	AI, Network Support
Target Audience	Mostly indies, coders	AAA-game studios, indies, artists
Coding	C#, Prefab, Bolt	C++, Blueprints
Community	More than 200k members on the official subreddit	About 100k members on the official subreddit
Performance	Does not scale well, unlike Unreal Engine	Has support for distributed execution (Incredibuild)

FIGURE 3.2: Comparison Table – Unity vs Unreal

3.1.1 Unity

Unity is a cross-platform game engine developed by Unity Technologies, first released in 2005. Initially launched as a Mac OS X-based game engine, Unity quickly gained traction due to its ease of use, flexibility, and ability to deploy across multiple platforms. Over the years, Unity has evolved into one of the most widely used game engines globally, powering a broad range of applications beyond just gaming, including simulations, architectural visualizations, automotive design, Augmented Reality (AR) and Virtual Reality (VR) development.

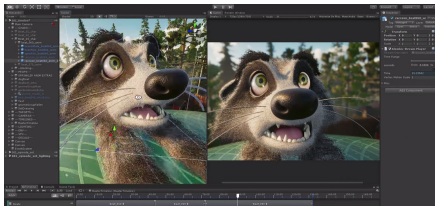
FIGURE 3.3: Unity

Unity's journey began at the 2005 Apple Worldwide Developers Conference, where it was introduced as a platform that democratizes game development by making it more accessible to independent developers. Unity's early versions supported 3D graphics for both desktop computers and mobile platforms, which helped position it as a leader in mobile game development, particularly for iOS and Android. By 2010, Unity 3.0 added critical features like deferred rendering and support for game consoles, allowing developers to create games with advanced graphical fidelity on both desktop and mobile platforms.

In 2015, Unity 5.0 was released, marking a significant leap in the engine's capabilities with features like real-time global illumination and a new audio system, making it even more attractive for AAA game studios and AR/VR developers alike. Unity's accessibility made it the go-to engine for beginners and professionals, with more than 1.3 million developers adopting its tools to create interactive experiences by 2015. It continued to push the boundaries, offering support for emerging platforms like Facebook's Oculus and Microsoft's Hololens 2.

Today, Unity is used across numerous industries due to its versatility:

- **Gaming:** Unity powers approximately 50% of mobile games and is known for its ability to deploy to multiple platforms seamlessly. Popular titles developed in Unity include Pokémon GO and Monument Valley.
- **Architecture and Engineering:** Many firms use Unity to create real-time 3D visualizations, which help stakeholders view and interact with virtual models of architectural designs.
- **Film and Animation:** Unity's Cinemachine feature allows filmmakers to create real-time animations, used in films like The Lion King remake, which blended real-time rendering with traditional filmmaking techniques.
- **Automotive Design:** Car manufacturers such as Volkswagen and BMW use Unity to create interactive simulations and visualizations for vehicle design and customer experience.



(A) Unity for Animation

(B) BMW in-vehicle experience

FIGURE 3.4: Unity in industries

Unity in AR Development

Unity has become one of the most popular platforms for developing Augmented Reality (AR) applications. Its AR Foundation toolkit enables developers to create applications that work seamlessly across multiple AR platforms, including Apple's ARKit, Google's ARCore, and Microsoft's Hololens 2. AR Foundation provides a unified workflow for developing cross-platform AR applications, allowing developers to build AR experiences for mobile devices and AR headsets without having to start from scratch for each platform.

When developing AR apps, Unity offers several advantages:

- **Cross-Platform Support:** Developers can write the code once and deploy it across different platforms, making it ideal for creating both mobile and head-worn AR applications.

- **Real-Time Rendering:** Unity excels in real-time 3D rendering, which is crucial for AR experiences that require responsive, interactive environments.
- **Integration with AR SDKs:** Unity's AR Foundation works as a bridge to integrate platform-specific SDKs like ARKit, ARCore, and Windows XR, making it easy to access features like spatial mapping, object tracking, and plane detection.
- **Customizability:** Unity provides a robust scripting environment using C#, allowing developers to customize every aspect of their AR applications.

For AR development, Unity supports a range of key features such as:

- **Spatial Mapping:** This allows virtual objects to be anchored to real-world surfaces, creating a more immersive and interactive experience.
- **Hand and Eye Tracking:** Essential for devices like Hololens 2, Unity supports advanced input methods such as hand gestures and eye tracking to interact with digital elements.
- **Environmental Understanding:** Unity provides tools for detecting planes, recognizing objects, and understanding light levels in the user's environment to adapt the AR experience accordingly.

For this thesis project, Unity was chosen as the primary game engine to develop the Hololens 2 application. Unity's powerful AR tools, such as AR Foundation and XR Interaction Toolkit, made it possible to integrate real-time GPS data, visualize flood hazards, and create an intuitive interface that first responders can use. The Universal Windows Platform (UWP) was used alongside Unity to deploy the application specifically for the Hololens 2, ensuring smooth compatibility with Windows-based devices.

3.2 Mixed Reality Toolkit (MRTK3)

The Mixed Reality Toolkit 3 (MRTK3) is the third generation of Microsoft's open-source toolkit designed to accelerate the development of cross-platform mixed reality applications. Initially built to support the Microsoft Hololens and Hololens 2, MRTK has since grown to support multiple platforms, including Android and VR devices. MRTK3 is deeply integrated with Unity and leverages the Unity XR Interaction Toolkit and OpenXR to provide developers with a comprehensive framework for creating immersive user interfaces (UI) and interaction systems.

3.2.1 MRTK's Evolution

The first version of MRTK was designed specifically for the first Microsoft Hololens, introducing new approaches to UX/UI design that were tailored to mixed reality devices. However, over time, the toolkit expanded in scope and was adapted to work on additional devices and platforms. MRTK2, which succeeded the original, offered a more modular architecture and extended support for cross-platform development, allowing developers to create applications for both Hololens and other AR/VR systems.

FIGURE 3.5: Mixed Reality Toolkit MRTK3

MRTK3, released in its latest form, brings further enhancements in terms of performance, modularity, and interaction paradigms. It supports dynamic scaling, volumetric UI, and improvements in 3D object manipulation, all while being highly optimized for resource-constrained platforms like the Hololens 2. In MRTK3, developers no longer need to duplicate efforts for different devices, as components created for one system (e.g., Hololens 2) can be easily adapted for other platforms, significantly reducing development time and complexity.

Key Features of MRTK3:

- **UI and Interaction Models:** MRTK3 provides an extensive library of pre-built UI components designed for 3D, volumetric interfaces. These components include hand-tracked menus, sliders, buttons, and toggle switches, all optimized for mixed reality. Unlike traditional 2D UI systems, MRTK3's UI elements are fully interactive in 3D space, allowing users to interact with virtual objects through gaze, hand tracking, and voice commands. For instance, users can manipulate objects using gestures like pinching or dragging, all while interacting with immersive UIs that are contextually bound to their environment.
- **Input and Interaction System:** The interaction system in MRTK3 supports multiple input methods, such as gaze-pinch, hand rays, and speech commands. The gaze-pinch interaction, for example, allows users to interact with virtual objects by simply looking at them and then pinching to select or manipulate. MRTK3 also supports motion controller inputs, making it flexible for applications that need to span both AR (such as Hololens 2) and VR devices. Additionally, the toolkit includes support for traditional 2D inputs like mouse and touchscreen, making it adaptable for both immersive and flat-screen applications.
- **Volumetric UI:** One of the standout features of MRTK3 is its support for volumetric UI, where UI elements are treated as real, physical objects in 3D space. This is particularly important for immersive applications like yours, where spatial mapping and interactions happen in real-world environments. Volumetric UIs provide a more intuitive interaction experience by allowing users to press buttons, manipulate sliders, and interact with menus that exist within the mixed reality space. The toolkit's data binding system ensures that these UI

elements remain responsive and dynamically updated, adapting to changing contexts or user inputs in real-time.

- **Optimized for Hololens 2:** MRTK3 is finely tuned for Hololens 2 and other resource-constrained devices. The toolkit minimizes per-frame memory allocation and is optimized for low-latency input processing, ensuring smooth performance even in complex environments. This is crucial for developing applications that require real-time responsiveness, such as emergency response applications, where users need immediate feedback from their interactions.

For this thesis, MRTK3 played a crucial role in building the user interface (UI) and interaction system for the Hololens 2 application. The toolkit's pre-built UI components (e.g., buttons, sliders, and hand menus) were essential in creating an intuitive interface that emergency responders can interact with hands-free. Additionally, MRTK3's gaze-based interactions and hand tracking allowed for seamless control of the AR elements in real-time, enhancing the user experience during critical tasks such as flood management and navigation in hazardous environments.

3.3 Microsoft Hololens 2

The Microsoft Hololens 2 is one of the leading devices in the field of mixed reality (MR), designed primarily for professional and enterprise use. Released in 2019, the Hololens 2 builds on the success of the original Hololens by offering improved ergonomics, expanded functionality, and advanced interaction capabilities, including hand tracking and eye tracking. It allows users to interact with virtual content that is seamlessly overlaid onto their physical environment, enabling hands-free operation in scenarios like data visualization, simulation, and remote collaboration.

FIGURE 3.6: Microsoft Hololens 2

Key Features of Hololens 2

- **Field of View (FoV):** The Hololens 2 offers a 52-degree diagonal field of view, significantly larger than the original model, providing users with more immersive holographic experiences. However, some users may still find the field of view somewhat limited when interacting with large-scale virtual objects.
- **Hand and Eye Tracking:** One of the standout features of the Hololens 2 is its advanced hand tracking and eye tracking systems. These allow users to interact with holograms using natural gestures such as pinching, grabbing, and dragging, without the need for physical controllers. Eye tracking adds another layer of precision, enabling users to interact with virtual content simply by looking at it.
- **Comfort and Wearability:** The device is designed for long-term use, with a balanced weight distribution achieved by placing the battery at the back of the head. This makes it comfortable to wear for extended periods. Additionally, the flip-up visor allows users to easily switch between augmented reality (AR) content and their real-world surroundings, enhancing usability in professional environments.
- **Display and Resolution:** The device offers a 1440x936 pixels per eye resolution, which provides clear and sharp visuals. However, the 60Hz refresh rate may not be ideal for fast-moving content, and the display performs best in indoor environments due to its challenges with outdoor lighting conditions.
- **Tracking and Spatial Mapping:** Hololens 2 features advanced spatial mapping and tracking, enabling precise interaction with virtual objects anchored in the real world. The device's tracking system ensures that holograms stay stable and realistic as users move around.

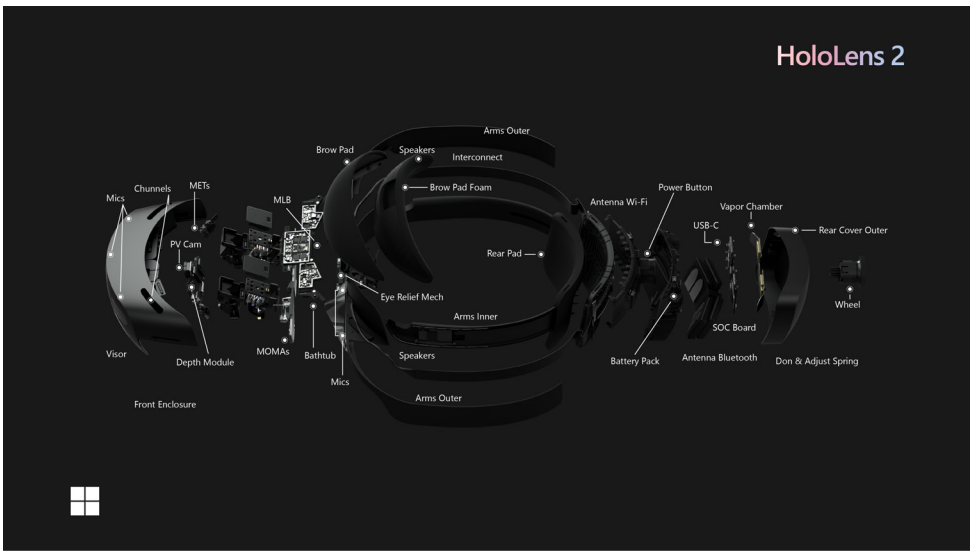


FIGURE 3.7: Hololens 2 Exploded View Diagram

Strengths of Hololens 2

- **Hands-Free Interaction:** The hand tracking, eye tracking, and voice command systems enable a completely hands-free experience, which is highly valuable

in fields like healthcare, manufacturing, and emergency response.

- **Enterprise Focus:** The device is optimized for enterprise use, integrating well with tools like Microsoft Dynamics 365 and offering robust features for remote collaboration, training, and simulations.
- **Long-Term Comfort:** The well-balanced design and flip-up visor make Hololens 2 comfortable for long periods of use, making it ideal for professionals who need to use the device during extended work sessions.

Limitations of Hololens 2

- **Field of View Limitations:** While the field of view is an improvement over the original Hololens, it remains limited compared to some other devices, which can affect the immersive experience when interacting with larger or more detailed holograms.
- **Outdoor Usability:** The device performs best indoors. In outdoor environments, particularly in bright light, the holograms may appear washed out or difficult to see, limiting a bit its use in outdoor scenarios.
- **Battery Life:** The Hololens 2 has a limited battery life, offering around 2-3 hours of continuous use, which may not be sufficient for long sessions without recharging.

In this thesis, Hololens 2 was used as the primary device for developing the AR-based flood emergency management system. The hand tracking and eye tracking capabilities of Hololens 2 were instrumental in creating a user-friendly interface, allowing first responders to interact with real-time data and visualizations while staying focused on their tasks. Additionally, the device's spatial mapping features enabled accurate placement of flood hazard indicators and visualizations, ensuring that responders could view critical information in their immediate environment as they navigated through flood zones.

3.4 Mapbox

Mapbox is a leading geospatial mapping platform that allows developers to create custom, interactive maps and integrate location data into their applications. Founded in 2010, Mapbox started as an open-source platform built on OpenStreetMap data, providing a customizable alternative to other proprietary mapping services. Over the years, Mapbox has evolved into a highly versatile platform, offering a variety of features like real-time geospatial data, geocoding, navigation, and Points of Interest (POI) visualization. Its flexibility and developer-friendly APIs have made it an essential tool across multiple industries, including logistics, gaming, and navigation systems.

Mapbox has grown significantly since its inception, originally catering to developers looking for open-source alternatives to proprietary map services. Its early focus was on providing customizable map experiences through open data. As demand for real-time location-based applications increased, Mapbox expanded its offerings to include turn-by-turn navigation, satellite imagery, and integration with real-time traffic data. Today, it powers location-based services for companies like

Uber, Snapchat, and Strava, making it a go-to solution for applications that rely on geospatial data.

Mapbox in Location-Based Applications

Mapbox has found a wide range of uses in location-based applications, where geospatial data is crucial for creating interactive, real-world experiences. By providing developers with tools to render maps, display Points of Interest (POIs), and integrate real-time location updates, Mapbox is used to power systems ranging from navigation apps to delivery platforms. For instance:

- **Logistics and Navigation:** Mapbox's real-time traffic and routing APIs are widely used in logistics and transportation, helping companies like Uber optimize routes and deliver better user experiences.
- **Gaming:** In location-based games like Pokémon GO, Mapbox helps developers create interactive environments where real-world locations are central to the gameplay. With POI data, these applications can dynamically update based on a player's physical location.
- **Event and Urban Planning:** City planners use Mapbox to create interactive maps for infrastructure planning, where real-time geographic data informs decision-making. These maps can display layers of information, such as public transportation routes, event locations, and public facilities.

FIGURE 3.8: Mapbox Applications and Maps

For this thesis, Mapbox was used within the Hololens 2 application to integrate a 2D map that provided real-time geographic data and Points of Interest (POIs) relevant to flood emergency management. Using Mapbox's Maps SDK for Unity, a 2D map

was rendered within the AR environment, allowing first responders to view critical locations directly in their field of view.

Key uses of Mapbox

- **POI Display:** Points of Interest such as hazard zones, rescue locations, and areas requiring assistance were integrated into the application using Mapbox's geospatial data. The dynamic POI updates provided real-time context, ensuring that first responders could act on the most current information.
- **Navigation:** Mapbox enabled real-time route calculations to assist responders in navigating flood-affected areas. Routes were visualized directly on the 2D map, overlayed in the AR interface for easy navigation without distractions.
- **Real-Time Updates:** The ability to dynamically update maps with new POI data ensured that responders were equipped with the latest information about hazards, making it easier to adapt to changing circumstances in the field.

Chapter 4

Server and Data Communication

4.1 Introduction

In any real-time augmented reality system, efficient and reliable data communication is crucial to ensure that users are provided with accurate, up-to-date information. For this project, a dedicated server is responsible for handling all data exchanges between the Hololens 2 devices, Kafka, and the mobile application that provides users' GPS data. The server enables seamless communication between these components, ensuring that critical information is transferred in real-time to support decision-making during flood emergencies.

The server plays several key roles in this system:

- **Real-time Data Transfer:** It manages the continuous flow of data between devices ensuring that information such as GPS location and hazard data is always current.
- **Kafka Integration:** The server acts as a bridge between the Kafka message broker and the Hololens 2 devices, ensuring that each device receives data from the Kafka topics in real time.
- **WebSocket Communication:** The server uses WebSocket technology to establish a real-time, low-latency communication channel between the mobile app, the Hololens 2 device and the server, enabling efficient transfer of data from the field.

This chapter will explore the architecture of the server, its integration with Kafka and WebSocket, and the methods used to manage and synchronize data between devices. The goal is to provide a comprehensive understanding of how the server supports the overall AR system and ensures that first responders have access to critical, real-time data in emergency scenarios.

4.2 Server Overview

The server is central to managing real-time data communication between the mobile application, Hololens 2 devices, and external data streams provided through Kafka. It is built using Node.js, a highly efficient, event-driven platform that allows for non-blocking I/O operations, which is critical for maintaining the real-time nature of this system. The server is responsible for managing the bi-directional flow of data between the mobile application and the Hololens devices, as well as receiving and

distributing essential information such as flood visualizations and real-time hazard data from Kafka.

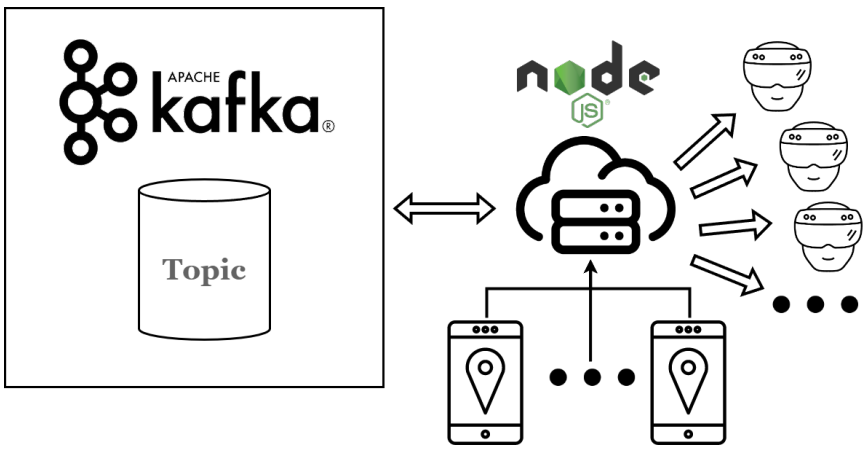


FIGURE 4.1: System Overview

One of the server's core functions is to handle WebSocket connections. WebSocket is a protocol designed for continuous, low-latency, two-way communication between clients and servers. In this system, WebSocket is used for two purposes: first, to receive GPS data from the mobile application in real time; and second, to forward that data to the appropriate Hololens devices. This setup allows for continuous data synchronization, ensuring that the Hololens devices display up-to-date location information, enabling first responders to act based on real-time GPS updates.

Another key responsibility of the server is to interact with Kafka for the transmission of real-time data streams. Kafka is a distributed event-streaming platform designed to handle high-throughput, real-time data feeds. In this project, Kafka is used to receive external data such as flood prediction visualizations, hazard occurrences etc. The server subscribes to a Kafka topic and consumes this data, which is then transmitted to the Hololens devices, allowing first responders to visualize critical information in the field.

In summary, the server orchestrates the flow of data between multiple devices and platforms, ensuring that first responders have access to both real-time GPS data and critical external updates such as flood predictions and hazard alerts. This seamless communication is essential for maintaining situational awareness and ensuring timely decision-making during flood emergency scenarios.

4.3 Data Flow and Communication Architecture

In this section, we will explore the various libraries and tools that were integrated into the server and how they contribute to real-time communication between the mobile application, Hololens 2 devices, and external data sources. Additionally, we will outline the overall data flow and communication architecture, highlighting how data moves through the system and the role each component plays in maintaining synchronization.

4.3.1 Libraries and Tools

Several important libraries and tools were used to implement the server, each selected for its ability to handle real-time data efficiently, manage connections, and ensure high performance in critical conditions. Below is a brief overview of each tool and its role within the system:

- **Node.js:** The server is built using Node.js, an asynchronous, event-driven JavaScript runtime. Node.js allows the server to manage multiple WebSocket connections and handle incoming requests with minimal delay, ensuring that real-time data can be processed and transmitted effectively.
- **WebSocket:** WebSocket is the protocol used for maintaining continuous, two-way communication between the mobile application and the Hololens devices. It allows the server to receive GPS data from the mobile app and forward it to the corresponding Hololens device in real time and also transmit key data recieved from kafka topic. WebSocket's low-latency communication is critical for ensuring that GPS updates reach the Hololens devices as soon as they are generated.
- **KafkaJS:** KafkaJS is the Node.js client library used to interact with Kafka, a distributed event-streaming platform. KafkaJS allows the server to subscribe to specific Kafka topics, receive messages in real time, and process the incoming data streams. In this project, KafkaJS is used to handle external data, such as flood predictions and hazard updates, which are consumed by the server and relayed to the Hololens devices.

```
// Create a new Kafka client
const kafka = new Kafka({
  clientId: "HoloLensDevice",
  brokers: brokers,
  logCreator: () => {
    return ({ namespace, level, label, log }) => {
      // Ignore log messages from KafkaJS
      if (namespace === "kafkajs") {
        return;
      }
      // Log other messages in a custom format
      const { message, ...rest } = log;
      console.log(`${label}: ${message}`, rest);
    };
  },
});
```

- **Express.js:** Express is the framework used to run the server and handle routing, HTTP requests, and WebSocket connections. It simplifies the management of endpoints and ensures that both WebSocket connections and Kafka subscriptions are managed efficiently. Express also provides the necessary infrastructure for the server to handle multiple requests and connections simultaneously.
- **JSON:** JSON is used to manage and process data, particularly for handling a CSV file containing flood level predictions over the next few hours. This file is converted to JSON format for easier processing and transmission within

the server. JSON is also used as the data format for exchanging information between the mobile app, the server, and the Hololens devices due to its lightweight and easily parseable structure.

4.3.2 Communication Pipeline

The data flow and communication pipeline orchestrated by the server is designed to manage real-time data transmission between multiple components, ensuring that both GPS data from the mobile app and external data streams are delivered to the Hololens devices. Below is an overview of how data moves through the system, highlighting the role of each component and how they work together to support continuous, low-latency communication.

User Connection to the Server

The first step in the data flow pipeline begins when the user launches the mobile application and connects it to the server. The mobile app establishes a WebSocket connection with the server sending a unique identifier in order for the server to see it as a mobile device connection.

```
wss.on("connection", async (ws, req) => {
  console.log("User-Agent:", req.headers["user-agent"]);

  let consumer;
  let phone = false;

  ws.on("message", (message) => {
    const rows = message.toString().split("\n");
    const firstRowSplit = rows[0].toString().split(",");

    const separatedStrings = message.toString().split(",");
    if (separatedStrings[1] === "ID") {
      //Create a new Kafka consumer for this WebSocket client
      consumer = kafka.consumer({ groupId: separatedStrings[0] });
      // Store the consumer in the map
      consumers.set(ws, consumer);
      clients[separatedStrings[0]] = ws;
    } else if (separatedStrings[0].includes("flood")) {
      if (separatedStrings[0].includes("1")) {
        console.log(1);

        ws.send(getValuesForTime(separatedStrings[1], c9r7Mapped) + ",water");
      } else {
        console.log(2);

        ws.send(getValuesForTime(separatedStrings[1], c8r8Mapped) + ",water");
      }
    } else if (separatedStrings[2] === "seek") {
      // When consumer connect to kafka topic, start seeking from specific offset
      consumer.seek({
        topic: currentTopic,
        partition: parseInt(separatedStrings[0]),
        offset: parseInt(separatedStrings[1]), //+1
      });
    } else if (message.toString() === "phone") {
      console.log("PHONE CONNECTED\n");
      phone = true;
    } else if (separatedStrings[2] === "gps") {
      // Send the location to the paired Unity device
      let unityClient = clients[separatedStrings[3]];
      if (unityClient && unityClient.readyState === WebSocket.OPEN) {
        unityClient.send(separatedStrings[0] + "," + separatedStrings[1] + "," + separatedStrings[2]);
      }
      console.log("Sending location to Unity client: %s\n", message.toString());
      sendToAll(message.toString());
    } else console.log("Received: %s", message);
  });
```

Identifier

Meanwhile, the Hololens 2 device must also connect to the server. Upon connection, the server sends a signal to the Hololens device, indicating the connection. The Hololens application then waits for the initialization signal from the server, that is

essentially the GPS location of the user, before starting to retrieve location information and set up the environment.

WebSocket Communication and GPS Data Transmission

Once both the mobile app and the Hololens are connected to the server, WebSocket is used for real-time transmission of GPS data. The server maintains separate WebSocket connections for the mobile app and each Hololens device. As the mobile app continuously generates GPS data, it is transmitted to the server via WebSocket, processed into a custom format for the unity application to identify.

```
if (separatedStrings[2] === "gps") {
        // Send the location to the paired Unity device
        let unityClient = clients[separatedStrings[3]];
        if (unityClient && unityClient.readyState === WebSocket.OPEN) {
          unityClient.send(separatedStrings[0] + "," + separatedStrings[1] + "," + separatedStrings[2]);
        }
        console.log("Sending location to Unity client: %s\n", message.toString());
        sendToAll(message.toString());
      }
```

The server then forwards the GPS data to the appropriate Hololens device using another WebSocket connection. This ensures that the Hololens device is continuously updated with the user's location, enabling real-time visualization of movement within the application.

Server and Kafka Integration

In addition to handling GPS data, the server is responsible for managing external data streams via Kafka. The server connects to a Kafka topic, via kafkaJS, that provides flood predictions and other important environmental updates. Once connected, the server uses KafkaJS to consume messages from the Kafka topic.

```
// Try to connect to Kafka
await consumer.connect();
// If the connection is successful, subscribe to the topic
await consumer.subscribe({ topic: currentTopic, fromBeginning: true });
```

The kafka consumer is the connected Hololens 2 device to the server. When a websocket connection to a device is set up, the server Binds the Hololens's device ID to a Kafka consumer. This way we seperate the consumers and also set up different group IDs for the topic connection.

```
ws.on("message", (message) => {
  const rows = message.toString().split("\n");
  const firstRowSplit = rows[0].toString().split(",");

  const separatedStrings = message.toString().split(",");
  if (separatedStrings[1] === "ID") {
    //Create a new Kafka consumer for this WebSocket client
    consumer = kafka.consumer({ groupId: separatedStrings[0] });
    // Store the consumer in the map
    consumers.set(ws, consumer);
    clients[separatedStrings[0]] = ws;
```

After receiving data from Kafka, the server processes this information and relays it to the Hololens devices through an established WebSocket connection. The Hololens devices then can visualize this data in the AR environment, alongside the real-time GPS data.

4.4 Synchronization and Reconnection

Synchronization between the mobile app, server, and Hololens devices is crucial to ensure that all data is transmitted accurately and without interruption. To maintain synchronization, the server must handle data from both WebSocket (GPS data) and Kafka (external data) seamlessly, making sure that all components stay in sync despite potential network issues.

One important feature of the system is its ability to handle connection retries. If a connection to Kafka or a WebSocket client fails or is temporarily lost, the server implements a retry mechanism to reconnect automatically. This ensures that temporary disruptions in communication do not result in data loss or prolonged downtime.

4.4.1 Connection Retry Mechanism

The retry mechanism continuously attempts to reconnect the server to the Kafka topic or WebSocket client in the event of a failure. As shown in the code below, the server attempts to reconnect to Kafka indefinitely until the connection is successfully established or until the WebSocket connection with the Hololens device is closed.

```
// This loop will keep trying to connect to Kafka indefinitely as long as the WebSocket client is connected
while (!phone) {
  try {
    // Try to connect to Kafka
    await consumer.connect();
    // If the connection is successful, subscribe to the topic
    await consumer.subscribe({ topic: currentTopic, fromBeginning: true });
    // Start consuming messages
    await consumer.run({
      // This function will be called for each message received
      eachMessage: async ({ topic, partition, message }) => {
        // Log the details of the message
        console.log("Received message:", {
          topic,
          partition,
          offset: message.offset,
          message: message.value.toString(),
        });
        //Send message to unity client
        ws.send(topic + "," + partition + "," + message.value.toString() + "," + message.offset + ",info");
      },
    });

    ws.send("Readys");

    // Read the JSON file

    // If we reach this point, the connection was successful, so we break the loop
    break;
  } catch (error) {
    // If an error occurs, log it and retry after a delay
    console.error("\nFailed to connect to Kafka, retrying...\n");
    await new Promise((resolve) => setTimeout(resolve, 1000)); // Wait for 1 second before retrying

    // If the WebSocket client has disconnected, stop trying to connect
    if (ws.readyState === WebSocket.CLOSED || phone) {
      //console.log('BREAK');
      break;
    }
  }
}
```

FIGURE 4.2: Kafka Connection Retry Mechanism

This ensures that the server can recover from temporary disconnections without losing critical data. This is particularly important when handling real-time data, such as GPS coordinates and flood prediction updates.

4.4.2 Data Synchronization

Once the WebSocket connection is re-established, the server resumes the transmission of data between the mobile app, server, and Hololens device. The same applies for Kafka—if the connection to the topic is lost, the retry mechanism ensures that the server resumes consumption of messages as soon as it reconnects, without losing the previously generated data, and that thanks to the nature of the kafka topic..

By maintaining continuous data synchronization and implementing a retry mechanism for connection handling, the server ensures that both GPS data from the mobile application and external data streams from Kafka are consistently delivered to the Hololens devices.

Chapter 5

Implementation

5.1 Introduction

This chapter will focus on the detailed implementation of the augmented reality (AR) system developed to assist emergency responders during flood events. The application was built in Unity, utilizing the Mixed Reality Toolkit 3 (MRTK3) for AR interactions on Hololens 2. The primary goal of the application is to provide real-time flood visualization and navigation tools, enhancing decision-making for first responders during critical rescue operations.

While the previous chapter covered the server-side components, including real-time data communication between the mobile application, Hololens devices, and external data streams, this chapter will dive deeper into the AR application itself. The application was designed to provide an intuitive interface for responders, enabling them to visualize crucial data such as flood predictions, maps, points of interest (POIs), and hazard warnings in real time.

The following sections will cover various aspects of the AR application's development, starting from the user interface (UI) design and interactions built in Unity to the water visualization techniques employed. Each section will provide insights into the tools, libraries, and methods used, along with the technical challenges encountered and solutions developed.

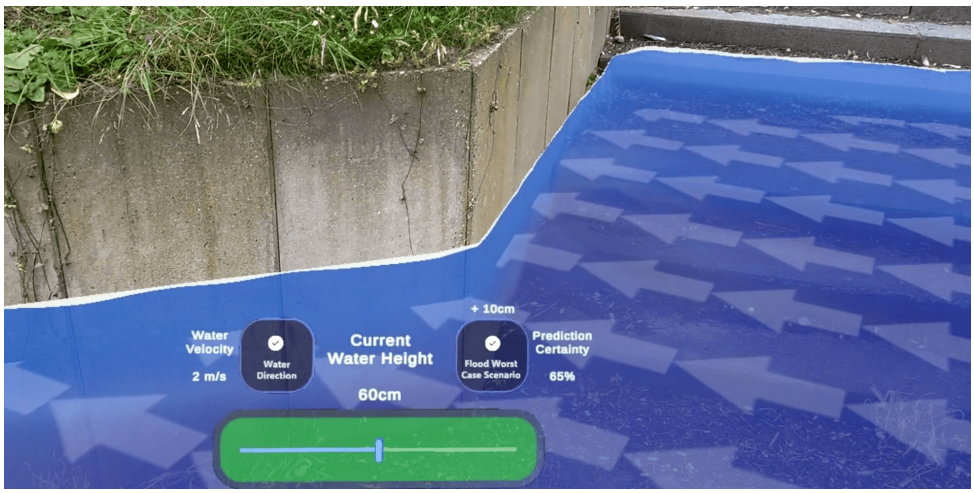


FIGURE 5.1: Water visualization in Dortmund trials

The key components of the application that will be explored in this chapter include:

- **User Interface (UI) Design:** A detailed look into the design and layout of the AR user interface. This section will cover the various menus, buttons, and interaction elements created to ensure an intuitive and user-friendly experience for first responders, balancing both simplicity and functionality.
- **Calibration Process:** The calibration process used within the application to ensure accurate alignment of the AR elements with the physical environment. This process is critical for providing precise visualizations that correspond correctly to real-world coordinates, enhancing user trust in the system.
- **Water Visualization:** Techniques and features used to visualize predicted flood levels, flow direction, and speed. This section will cover the structure, including how the application visualizes the water level predictions, components used and general approach.
- **Points of Interest (POIs):** The design and implementation of POIs, which highlight critical locations within the AR environment. POIs include features such as dynamic updates based on risk level, dynamic positioning, visual indicators for navigation and prioritization to help responders focus on areas requiring immediate attention.
- **Interactive Map (Mapbox Integration):** An interactive map integrated using Mapbox, which allows responders to view their location, navigate to points of interest, and visualize key areas affected by flooding. This map serves as a critical navigational aid, providing responders with a real-time view of their surroundings.

5.2 System Architecture

The system is designed to provide real-time augmented reality (AR) visualizations and navigation tools to assist first responders during flood emergencies. The architecture consists of several key components, each playing a specific role in ensuring that the system delivers accurate, real-time data to the Hololens 2 devices.

Overview of the Architecture

As shown in Figure 5.2, the architecture is centered around the interaction between the mobile application, the server, and the Hololens 2 AR application, with external data sources such as Mapbox for interactive maps and Kafka for real-time flood forecasting and hazard data.

- **Mobile Application:** The mobile app captures the user's GPS data and sends it to the server using WebSocket. This provides continuous location tracking, which is critical for real-time navigation and positioning within the AR environment.
- **Server:** The server acts as the central hub, receiving GPS data from the mobile app and external flood prediction data from Kafka. It then transmits this data to the Hololens devices via WebSocket. The server ensures that the data reaches the AR application in real time, allowing for continuous updates in the AR environment.

- **Hololens 2 AR Application:** The AR application on the Hololens 2 device visualizes GPS data, flood predictions, and other critical information. One of the key processes handled within Unity is the translation of real-world GPS coordinates into the Unity local coordinate system, allowing accurate positioning of AR elements such as points of interest (POIs), hazards, and other visual markers.
- **Mapbox Integration:** The application uses Mapbox for real-time map visualizations, allowing responders to view their location and navigate through the environment using a map-based interface. Mapbox helps visualize GPS data in 2D/3D map formats, aiding navigation and decision-making in the field.

The diagram below illustrates the architecture of the system, showing the flow of data between the mobile app, server, Hololens, and external data sources.

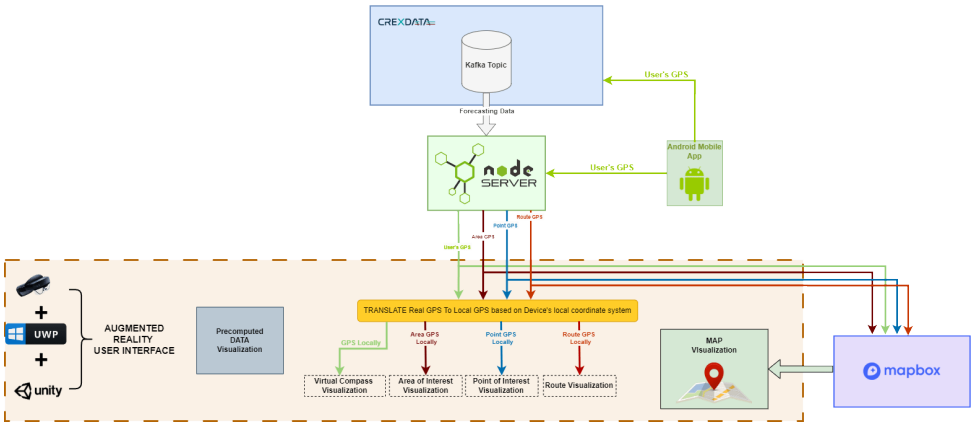


FIGURE 5.2: System Architecture

This architecture ensures that all data flows seamlessly between the components, enabling first responders to access real-time, accurate information to make informed decisions during flood emergencies.

5.3 User Interface (UI)

The design of the user interface (UI) for the augmented reality (AR) application was driven by the need for a simple, intuitive, and accessible layout that first responders can quickly navigate. The primary goals were to provide access to critical functionalities without overwhelming users with clutter or unnecessary options. Key design considerations included:

- **Usability in AR:** Designing a UI that complements the limited field of view on Hololens 2 and avoids cluttering the user's vision while still presenting essential information.
- **Accessibility and Responsiveness:** Creating a UI that can be quickly accessed and easily operated, even in stressful scenarios. Fast navigation and ease of interaction were prioritized to ensure the application could be practical in real-world emergency situations.

Initial UI design iterations revealed that while the overall layout was functional, some first responders found certain menu elements difficult to use, particularly the

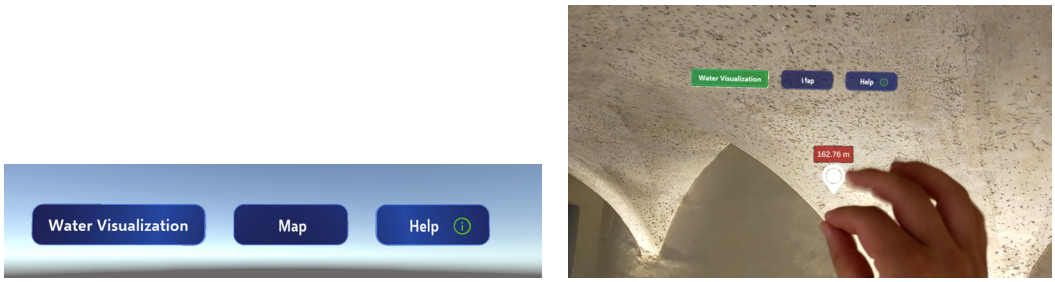
initial menu design that was hard to get comfortable with, especially for an untrained user on this technology. Feedback from field trials led to iterative improvements in the UI, resulting in a more intuitive, layered design that made interacting with the system faster and more accessible.

5.3.1 UI Components and Layout

The current version of the main menu is positioned to be easily accessible via the palm-up gesture (Figure 5.4), meaning users can activate the menu by facing their left palm toward the Hololens. This makes the menu feel like a part of the user's body, providing an intuitive method of interaction. The redesigned menu has proven to be significantly more user-friendly compared to the initial version, as observed during follow-up trials with first responders.

Initial Menu Design

The initial version of the menu featured a more complex layout that required users to perform multiple gestures and movements to access basic functionalities, such as aiming and ray pointing to access the functionalities. Although the initial design placed the menu above the user's head and kept it visible at all times (Figure 5.3a), this proved challenging for untrained users, where first responders struggled to quickly navigate the menu and use key features effectively as it needed to aim and point to select features and this was not working appropriately at all times (Figure 5.3b).



(A) (B)

FIGURE 5.3: Initial Menu Design

Redesigned Menu

In response to user feedback, the menu was simplified, resulting in the current design, which features large, easily accessible buttons and a clear layout. The menu contains options for enabling key features, each of which is explained below:

- **Hand Rays Toggle:** This option allows users to enable or disable hand rays, which are used to interact with objects at a distance. Disabling hand rays helps reduce distractions when interacting with nearby objects.
- **Water Visualization Toggle:** Activates or deactivates the flood visualization feature, which shows real-time water levels and flow direction in the AR environment.

- **Map Toggle:** This option enables or disables the interactive map feature, providing users with a 2D/3D view of their location and surrounding points of interest (POIs) using Mapbox.
- **Help:** Opens a help menu that provides guidance on how to use the application and its features.
- **POI Filter:** Allows users to filter points of interest based on specific categories, such as schools or hospitals. This feature helps responders focus on the most relevant locations during an emergency.
- **Calibrate:** Initiates the calibration process, guiding the user through aligning the AR environment with real-world GPS coordinates. A new guide window opens once this option is selected, providing step-by-step instructions for calibration.

(A) (B)

FIGURE 5.4: Improved Menu Design

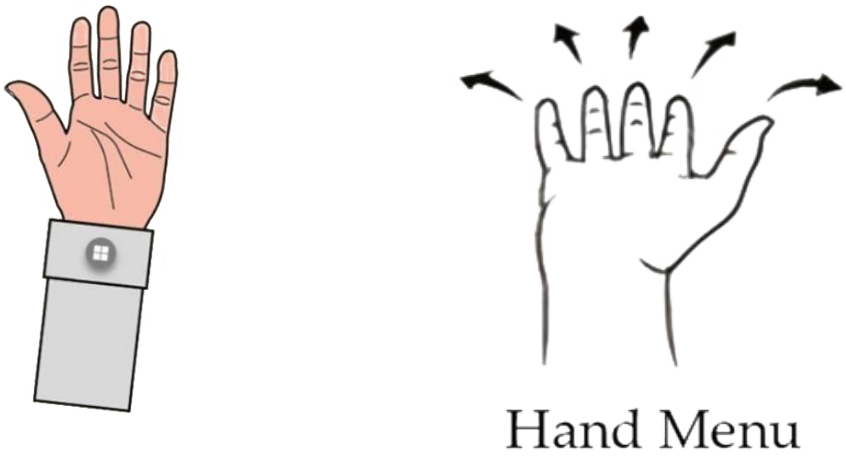
Through these improvements, the application now features a more intuitive and user-friendly menu, making it easier for first responders to interact with critical features during emergency operations.

5.3.2 Interaction Methods

Interaction within the application leverages a combination of manual input (finger pressing buttons), gaze tracking, and gestures to provide users with a flexible and intuitive way to navigate and control the AR interface. The Mixed Reality Toolkit 3 (MRTK3) in Unity was used to implement these interactions, allowing for smooth and responsive controls tailored to the Hololens 2.

Gesture Interaction

The menu can be accessed by raising the user's left palm toward the Hololens device, which triggers the recognition of the palm gesture using MRTK3. The menu is then dynamically positioned and tracked based on the user's hand position, ensuring that it stays in view but remains unobtrusive.



(A) (B)

FIGURE 5.5: Hand Palm Recognition Motion

Manual Interaction

Users interact with the menu by pressing options with their finger (manual input). This straightforward interaction method ensures that key functionalities like enabling the map or toggling water visualization are easily accessible. Each menu item is activated through direct selection, with no need for complex gestures.

(A) (B)

FIGURE 5.6: Manual Input Interaction (Finger Pressing)

Gaze Interaction

Gaze interaction is a key method for hands-free control in the application, allowing users to select and interact with elements such as Points of Interest (POIs) by simply looking at them. The Mixed Reality Toolkit 3 (MRTK3) provides the underlying framework that handles gaze tracking on the Hololens 2, making this interaction smooth and responsive.

With gaze tracking, the system continuously monitors the user's eye movements and focuses on where the user is looking within the AR environment. MRTK3's gaze tracking system automatically locks onto the UI element (such as a POI) when

the user focuses on it, highlighting or selecting the element based on gaze duration or additional confirmation actions (such as a simple dwell or manual confirmation).

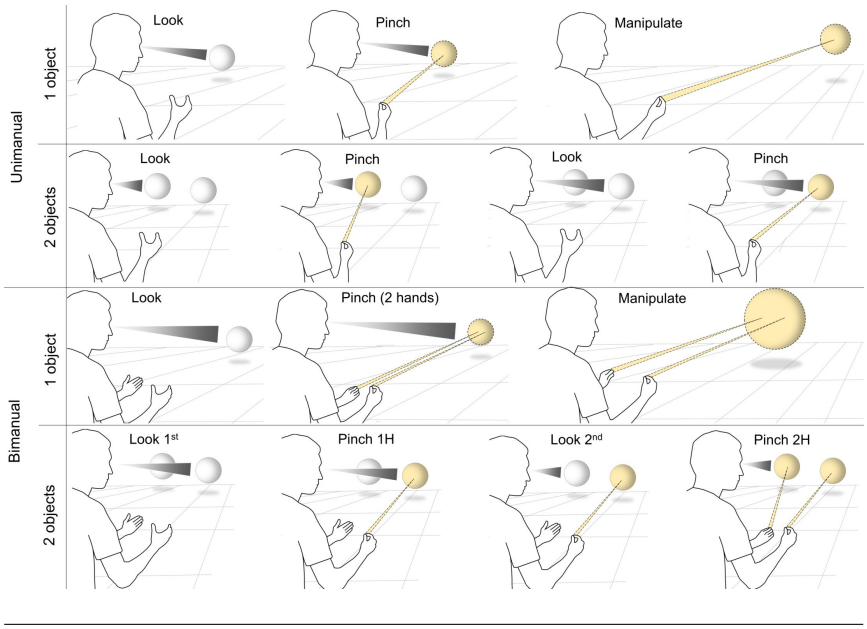


FIGURE 5.7: Gaze Principle

MRTK3 includes predefined gaze-based interaction profiles that simplify the development process. It allows to easily assign gaze behaviors to various elements, meaning that as soon as the user locks their gaze on an interactive object, the object reacts accordingly. This can involve highlighting the element, revealing more information, or triggering an action.

By using MRTK3 to manage gaze tracking, the application leverages Hololens 2's built-in eye-tracking technology to provide accurate, seamless, and efficient hands-free interaction.

Finger and Gaze Interaction

Certain UI elements, such as the water slider (which will be discussed in a later section), allow for both finger input and gaze-based interaction. This dual-method approach gives users flexibility in how they choose to interact with the application. For example, when fine-tuning visualizations like flood levels, users can either drag the slider with their finger or use gaze tracking for more hands-free control.

By combining different interaction methods through MRTK3, the system provides a robust and adaptable user experience that suits the fast-paced and dynamic nature of emergency response situations.

5.3.3 Integration of MRTK3 for Interaction and UI Elements

The Mixed Reality Toolkit 3 (MRTK3) played a pivotal role in implementing interaction controls and user interface elements within the AR system on Hololens 2. MRTK3 provides a robust set of components that simplify the development of AR interfaces, allowing for quick and accurate interaction with both 2D and 3D objects in the environment.

Using MRTK3 for Hand Interaction

The Hand Constraint Palm Up component from MRTK3 was used to create the gesture-based menu activation. This component tracks the position of the user's palm and enables the dynamic appearance of the menu when the palm is raised toward the device camera.

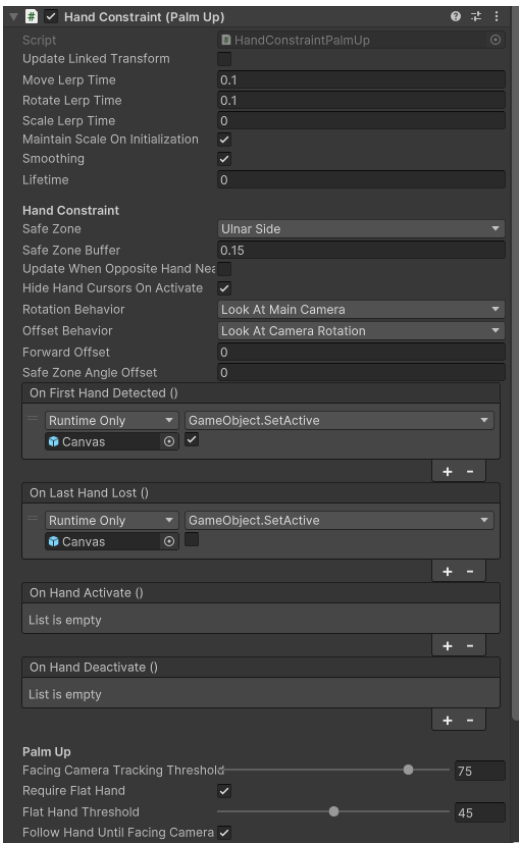


FIGURE 5.8: Hand Constraint (Palm Up) Component

Solver Handler was utilized in many situations but here was used to ensure the menu's position remains locked in relation to the user's hand, providing a seamless interaction experience. As the user moves their hand, the solver ensures that the menu tracks the hand movement smoothly without obstructing the user's field of view.

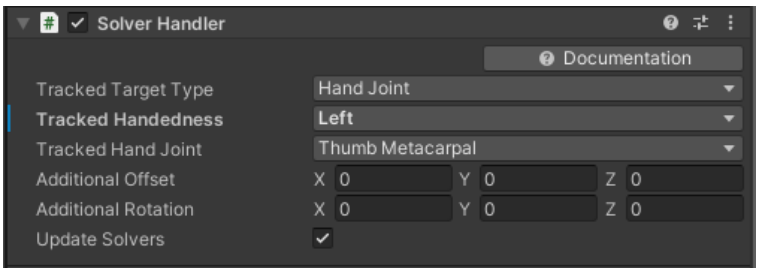


FIGURE 5.9: Solver Handler Component

Implementing Gaze Interaction with MRTK3

MRTK3 includes built-in support for gaze tracking on Hololens 2. Gaze-based interactions were implemented to allow users to interact with objects such as POIs by simply looking at them. By leveraging MRTK3's gaze interaction profiles, the application automatically highlights objects when the user's gaze is fixed on them, ensuring precise interaction even without manual input.

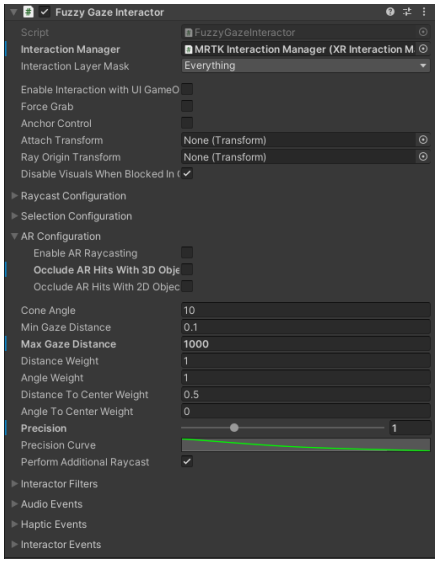


FIGURE 5.10: Gaze Interactor Component

Menu and UI Component Integration

MRTK3 provides several UI prefabs and interaction controls that were customized for this application. Components like buttons, sliders, and dialog boxes were adapted to the AR environment using MRTK3's flexible toolkit.

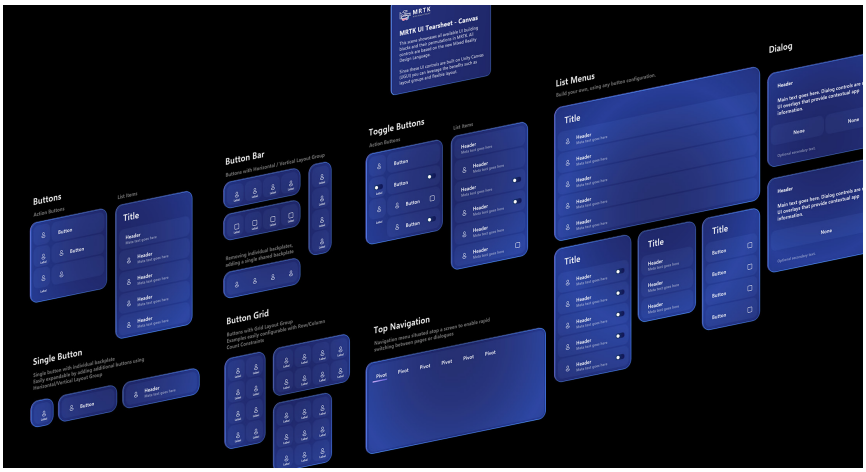


FIGURE 5.11: MRTK 3 Example Prefabs

For example, the Hand Rays Toggle in the main menu was built using MRTK3's interactive toggle prefab, which is optimized for hand interaction. Similarly, other buttons and sliders were designed using MRTK3's responsive UI components.

5.4 Calibration Process

The calibration process is a critical step at the start of the AR application to ensure that the virtual elements are correctly aligned within the real-world environment. The goal is to accurately position augmented reality objects, such as Points of Interest (POIs), maps, and flood visualizations, in relation to the user's real-world position and orientation.

Without proper calibration, the AR environment's directional references (like North) would not match the real world, leading to misaligned elements, which could hinder first responders' ability to navigate accurately.

5.4.1 Problem Statement

A key issue faced during development was that Unity's North (the virtual environment's default orientation) did not align with true North in the real world. Upon application startup, the user is placed at position (0, 0, 0) in the Unity AR environment, facing the environment's default North. However, this was often misaligned with the real-world North, leading to a disconnection between the user's actual orientation and the AR system's virtual North.

This mismatch meant that, without calibration, the AR system's directional features like POI placement and live danger popups would be inaccurately positioned in the real world, reducing the effectiveness of the application for field operations.

5.4.2 Calibration

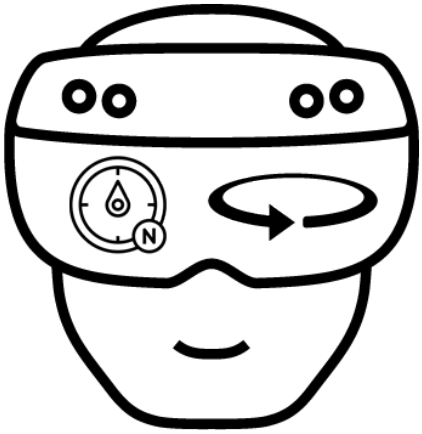
To solve this problem, a calibration process was introduced which ensures the AR system's North aligns with true North in the real world. This calibration process occurs after the application starts and can be initiated again at any time through the Hand Menu if the environment seems off. By completing the calibration, the system ensures that all AR objects are accurately placed and oriented in the environment.

The process requires the user to follow a set of instructions (as seen in the Calibration Instructions UI), which guide them to face true North before pressing the Calibrate button. At this point, the system runs the Align() function, which adjusts the rotation of the AR environment to match the user's actual orientation.

FIGURE 5.12: Calibration Interface

5.4.3 How It Works

The user first is prompted to open a compass app on their phone, align it to face true North, and then face that direction while wearing the Hololens 2 device. The system displays a set of instructions on the screen to guide the user through this process.



Once the user is properly aligned with true North, they press the Calibrate button, which triggers the calibration function in the application. This ensures that the virtual environment's directional reference is aligned with real-world North.

When the Calibrate button is pressed, the **Align** function is called. This function calculates the difference between the user's current rotation and the correct orientation (true North), then applies this rotation to align the virtual environment accordingly.

```
[SerializeField] private GameObject _objectForAlignment; //The object that will be aligned (The whole scene is MYORIGIN object)
[SerializeField] private Camera _mainCamera;
private int _deegrees;
public void Align()
{

    //Calculate the angle between the given deegrees and the main camera
    float angle = _objectForAlignment.transform.rotation.eulerAngles.y - _mainCamera.transform.rotation.eulerAngles.y;
    Debug.Log("Angle: " + angle);
    //Change the align objects Y rotation to the calculated angle
    _objectForAlignment.transform.rotation = Quaternion.Euler(0, angle, 0);
    this.gameObject.SetActive(false);
```

FIGURE 5.13: Alignment Method

After the process is done, the entire AR environment, including the virtual POIs, map, and flood visualizations, is rotated so that its North now matches the real-world North. This ensures that all AR elements remain correctly aligned as the user moves through the environment. This results to a typical offset of about 4 to 5 degrees, which is the average and acceptable error due to human factors.

5.5 Augmented Reality Map (Mapbox Integration)

This section will explore the implementation of the interactive map in the augmented reality application, which plays a critical role in enhancing users' navigation and situational awareness. The map provides real-time visualizations of the user's location and Points of Interest (POIs), helping first responders navigate efficiently in emergency scenarios.

The map is built using the Mapbox SDK, which integrates seamlessly with Unity to provide dynamic, real-time data. The map offers essential features such as displaying the user's GPS location, showing nearby POIs, and allowing for customized interactions.

FIGURE 5.14: Augmented Reality Map

In this section, we will cover

- How Mapbox was integrated into Unity using the Mapbox SDK and the API key configuration.
- The interaction methods, including how users can enable or disable the map via the hand menu.
- The features offered by the map, including real-time user location tracking and dynamic POI visualization.
- A technical breakdown of how these features were implemented, with a focus on the scripts and methods used to manage the map's data flow and updates.

5.5.1 Using Mapbox SDK in Unity

To bring mapping capabilities into the augmented reality environment, Mapbox SDK was integrated into Unity. Mapbox provides the flexibility and tools necessary to create a customized map that updates based on real-world data, making it ideal for AR applications like this one.

Integration Process

The process of integrating Mapbox SDK into Unity involves:

- **Installing the Mapbox SDK:** The SDK can be downloaded and imported into Unity by downloading it directly from the Mapbox website. Once installed, Mapbox provides components and prefabs that can be easily added to the Unity scene.
- **Setting up the Mapbox Map:** After installing the SDK, a Mapbox map prefab is added to the scene. This prefab serves as the foundation for displaying maps

and includes a set of customizable options, such as map style, zoom level, and data layers.

- **API Key Usage:** In order to access Mapbox services, an API key must be obtained and added to the project. This key allows the application to authenticate with Mapbox's servers and pull in the necessary map data. The API key is managed within Unity by adding it to the Mapbox configuration panel.

Mapbox Features in the AR Application

Using Mapbox SDK, the AR application renders a 2D map that includes:

- User's real-time location based on GPS data.
- POIs displayed as interactive markers.
- Custom styling to ensure clarity and usability in the AR environment.

5.5.2 Enabling the Map

In the augmented reality application, the map feature is controlled via the hand menu. This ensures that the map can be toggled on or off based on the user's immediate needs. This clean and easy interaction method allows first responders to quickly access the map without disrupting their workflow.

Interaction Method

The map can be enabled or disabled by selecting the Map Toggle button on the hand menu (Figure 5.15). The hand menu appears when the user performs a palm-up gesture, which is detected by MRTK3 and triggers the menu to appear. The user can then use finger selection to press the Map Toggle button, activating or deactivating the map view within the AR interface.

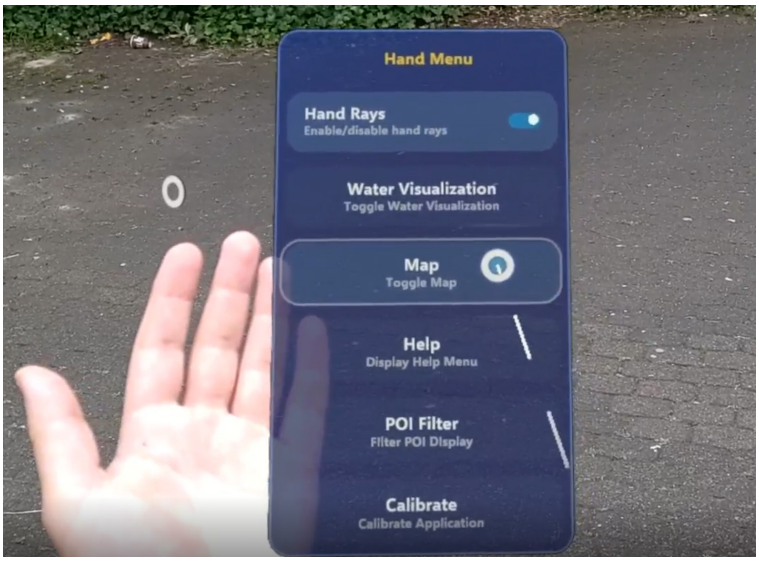


FIGURE 5.15: Enabling Map

When the map is enabled

The map is being loaded through the MapBox API at the user's current location which is also displayed on the map. Any nearby Points of Interest (POIs) are shown as markers on the map, providing the user with key information about important locations.

5.5.3 Map Features

The interactive map in the AR application provides essential real-time information and key features to assist first responders during flood emergencies.

User Location

The map displays the user's real-time GPS location provided by the Android application through the server. As the user moves, the map updates continuously (every 5̃ seconds), ensuring the displayed location remains synchronized with the user's actual movements.

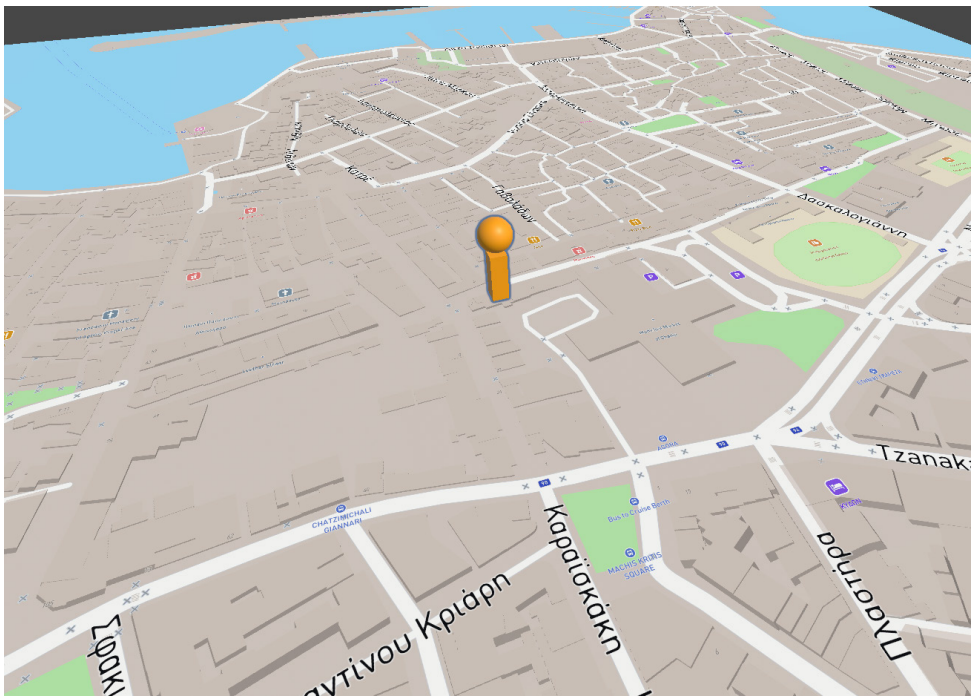


FIGURE 5.16: User's Location Display

Points of Interest (POIs)

The map also shows Points of Interest (POIs) such as hospitals, schools, and hazard zones, marked with distinct icons. These POIs are dynamically updated and superimposed based on their real-world GPS coordinates, providing responders with a visual guide to critical locations in their vicinity.

FIGURE 5.17: Points of Interest (POIs) on the Map

Filtering POIs

A unique feature of the system is the ability to filter POIs through the hand menu. By using the POI Filter option, responders can choose to display or hide specific categories of POIs based on their needs. For example, they may filter out non-critical locations like schools to focus on more relevant areas such as hospitals or hazard zones. The POI filtering feature applies not only to the map but also to the 3D AR environment.

(A) (B)

FIGURE 5.18: Filter Menu Interface

5.5.4 Technical Implementation

The map offers several features that enhance navigation and situational awareness, all of which were implemented using the Mapbox SDK and custom Unity scripts. Below is a detailed breakdown of how these features were technically implemented.

User Location Tracking

The user's real-time location is updated on the map through a combination of GPS data transmitted from the Android mobile application and a WebSocket connection to the server. The user's current latitude and longitude coordinates are sent to the Hololens device, and these values are processed and reflected on the Mapbox-powered map.

```
if (separatedStrings[2] === "gps") {
        // Send the location to the paired Unity device
        let unityClient = clients[separatedStrings[3]];
        if (unityClient && unityClient.readyState === WebSocket.OPEN) {
          unityClient.send(separatedStrings[0] + "," + separatedStrings[1] + "," + separatedStrings[2]);
        }
        console.log("Sending location to Unity client: %s\n", message.toString());
        sendToAll(message.toString());
      }
```

FIGURE 5.19: Server side GPS transmission

Receiving GPS Data

The Hololens application receives the GPS data from the server in the form of messages. The HandleMessage() function is responsible for processing these messages. When a message containing GPS coordinates is detected, the message is split into latitude and longitude values, which are then passed to the mapManager to update the user's position on the map.

```

void HandleMessage(string message)
{

    //format the message
    string rawMessage;
    try
    {
        rawMessage = RawMessage(message);
    }
    catch (Exception ex)
    {
        Debug.LogWarning("An error occurred when formatting the message: " + ex.Message);
        rawMessage = message;
    }
    //Debug.Log(message);

    if (message == "Readys")
    {
        Debug.Log("Ready!");
    }

    if (message.Contains("gps"))
    {
        double latitude = double.Parse(message.Split(',')[0], CultureInfo.InvariantCulture);
        double longitude = double.Parse(message.Split(',')[1], CultureInfo.InvariantCulture);

        spawner.UpdateReferenceCoordinates(latitude, longitude);

        mapManager.SetCoordinates(new Vector2d(latitude, longitude));
        mapManager.UpdatePlayerLocation(new Vector2d(latitude, longitude));
    }
```

FIGURE 5.20: User's GPS Handling section

Updating the Map

Once the latitude and longitude values are received, the UpdatePlayerLocation() function is called within the mapManager. This function uses Mapbox's built-in methods to convert the real-world GPS coordinates into Unity world coordinates. The map is updated to center the user's location, ensuring that it accurately reflects the user's current position.

```
public void UpdatePlayerLocation(Vector2d coordinates)
{
    if (!mapInitialized) return;

    try
    {
        if (MapGameobject.activeSelf)
        {
            map.SetCenterLatitudeLongitude(coordinates);
            map.UpdateMap();
            Player.transform.position = map.GeoToWorldPosition(coordinates);
            Debug.Log("Player location:" + Player.transform.position);
        }
    }
    catch (System.NullReferenceException)
    {
        Debug.LogWarning("Map is not initialized yet");
    }
}
```

FIGURE 5.21: Update Users Location function

Key Steps in the Update Process

- The `SetCenterLatitudeLongitude()` method from Mapbox sets the center of the map to the user's updated location for better map visibility.
- The `GeoToWorldPosition()` method converts the received latitude and longitude into Unity's world coordinate system.
- The user's `transform.position` in the Unity scene is updated to match the converted world position, effectively moving the user's avatar on the map.

Points of Interest (POIs) Map Implementation

The Points of Interest (POIs) are critical for providing first responders with important location-based information such as hospitals, schools, or hazard areas. These POIs are dynamically spawned on the map as markers, based on real-world GPS coordinates received from a Kafka topic through a WebSocket connection.

Receiving and Handling POI Data

The server acts as a Kafka consumer, continuously reading messages related to POIs from a Kafka topic. When the server receives a message that includes POI data, it forwards this message to the Hololens application through the WebSocket connection. In Unity, the message is processed and parsed to identify whether it contains a command to spawn a new POI.

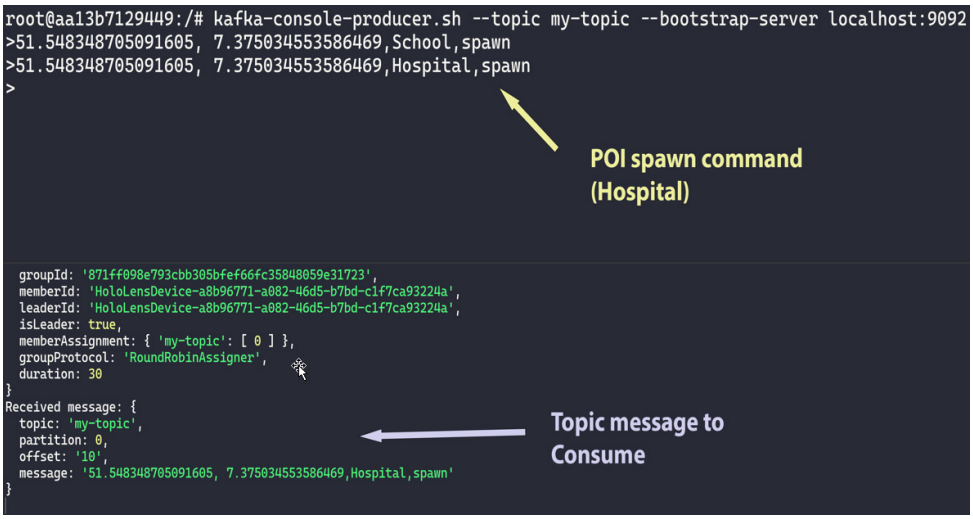


FIGURE 5.22: Kafka Example POI spawn string

```
await consumer.run({
  // This function will be called for each message received
  eachMessage: async ({ topic, partition, message }) => {
    // Log the details of the message
    console.log("Received message:", {
      topic,
      partition,
      offset: message.offset,
      message: message.value.toString(),
    });
    //Send message to unity client
    ws.send(topic + "," + partition + "," + message.value.toString() + "," + message.offset + ",info");
  },
});
```

FIGURE 5.23: Server topic message transmission

Message Processing

As shown in Figure 5.24, Unity listens for the POI spawn command. When a message containing the keyword spawn is received, it is processed to extract the relevant details, including the POI's name, description, type, and coordinates.

```
if (rawMessage.Contains("spawn"))
{
    Debug.Log("Spawn section handling: " + rawMessage);
    // This assumes that the latitude and longitude are part of the 'raw_message'
    string[] spawnParts = rawMessage.Split(',');
    //check if spawnParts is the correct length
    if (spawnParts.Length < 3)
    {
        Debug.LogWarning("Invalid spawn message: " + rawMessage);
        return;
    }
    string spawnString = spawnParts[0] + "," + spawnParts[1] + "," + spawnParts[2];
    Debug.Log("Spawn string: " + spawnString);

    if (spawner != null)
    {
        try
        {
            spawner.SpawnPOIFromString(spawnString);
        }
        catch (Exception ex)
        {
            Debug.LogError("An error occurred when spawning the object: " + ex.Message);
        }
    }
    else
        Debug.Log("Spawner is null");
}
```

FIGURE 5.24: Application Spawn Message Handling

Spawning POIs on the Map

Once the POI data is received and processed, the SpawnPOIFromString() method is called to handle the placement of the POI in the AR environment. The method itself calls the SpawnPOIForMap() which first checks if the reference coordinates are set, and then instantiates a new POI GameObject on the map, based on its real-world GPS coordinates (Figure 5.25).

```
public void SpawnPOIForMap(GPSCoordinate coord)
{
    if (referenceCoordinatesSet)
    {
        try
        {
            GameObject prefab = mapPrefabs[coord.POIType];
            //spawn the object as child of mapGameObject
            GameObject createdObj = Instantiate(prefab, mapGameObject.transform);
            //Set created object's position to 0,0,0
            createdObj.transform.localPosition = new Vector3(0, 0, 0);
            //Add to mapObjects list of the mapManager
            mapGameObject.GetComponentInParent<MapAndPlayerManager>().spawnedPOIs.Add(createdObj, new Vector2d(coord.Latitude, coord.Longitude));
            //Update the spawned POIs on the map
            mapGameObject.GetComponentInParent<MapAndPlayerManager>().UpdateSpawnedPOIs();
        }
        catch (Exception ex)
        {
            Debug.LogError("Failed to spawn map point object: " + ex.ToString());
        }
    }
    else
    {
        Debug.LogError("Reference coordinates not set");
    }
}
```

FIGURE 5.25: Spawn POI on the minimap

The SpawnPOIForMap() method chooses a prefab, based on the POI type, from the prefab pool list and instantiates it at the default position on the map. The POI prefab is then added to the list of active map POIs, ensuring that it can be updated or

removed as needed. Then the UpdateSpawnedPOIs() method is called on the MapManager to update the POIs position on the map to correctly reflect their real-world coordinates (Figure 5.26).

```
public void UpdateSpawnedPOIs()
{
    if (!mapInitialized || !MapGameobject.activeSelf) return;

    foreach (KeyValuePair<GameObject, Vector2d> poi in spawnedPOIs)
    {
        poi.Key.transform.position = map.GeoToWorldPosition(poi.Value);
        poi.Key.transform.position = new Vector3(poi.Key.transform.position.x, poi.Key.transform.position.y + 0.03f, poi.Key.transform.position.z);
    }
}
```

FIGURE 5.26: Update POIs Map Position

This method iterates through all spawned POIs, updating their position whenever necessary to maintain correct alignment with the map as the user moves or as new data is received.

POI Details

Each POI is created using the GPSCoordinate class, which contains detailed information such as the POI's name, description, type, and risk level. This allows the system to display the POI both on the map and within the AR world, providing users with a fully integrated experience. This will be covered more on the dedicated section about Points of Interest (POIs).

5.5.5 Map Interaction and Placement

The map is designed to be interactive and adaptable to the user's needs. It is positioned just below the user's chest area to ensure it doesn't obstruct the field of view during navigation. This allows the user to glance down to access the map while maintaining full situational awareness in front of them.

The map can also be grabbed and manipulated (Figure 5.27) to provide different viewing angles, enabling the user to get a better perspective of the environment. Users can move or rotate the map to get the optimal angle for navigating or viewing important points of interest. This level of interaction ensures that the map remains both functional and non-intrusive during real-time emergency response scenarios.

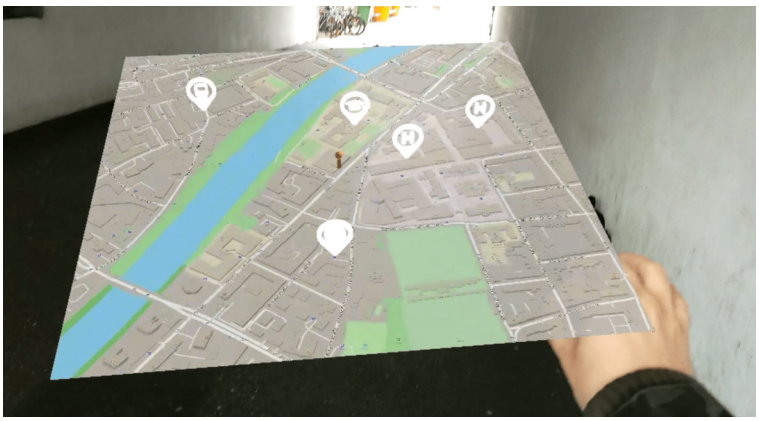


FIGURE 5.27: Map Interaction

5.5.6 Conclusion

In summary, the map provides a set of features that assist first responders in navigating complex environments during flood emergencies. Through real-time user location updates and Points of Interest (POI) placement, the map offers critical situational awareness and general understanding and view of the surrounding area. The interactive design allows users to manipulate the map for better perspectives while keeping it conveniently placed to avoid obstructing their vision. These functionalities make the map a critical tool in the AR application, offering navigation and virtual information overlays.

5.6 Points of Interest (POIs) in the AR Environment

Points of Interest (POIs) are critical elements in the augmented reality environment that guide first responders to key locations such as hospitals, shelters, or areas of high risk. These POIs provide essential location-based information in real-time, displayed as virtual markers within the AR environment.

In this section, we will explore the technical implementation of POIs within the system. The following subsections will cover:

- The design and structure of POIs, including the prefabs used in Unity.
- How POIs are dynamically spawned into the AR environment and how they are correctly placed based on GPS Coordinates.
- Other advanced features and customization options for POIs to enhance user experience and situational awareness.

5.6.1 POI Design and Structure

The design of Points of Interest (POIs) is crucial in enhancing situational awareness for first responders during flood emergencies. Each POI is represented by unique 3D models, designed to convey critical information at a glance. The models were created using Blender, a powerful open-source 3D modeling software, which allows for intricate designs and textures.

In Blender, I crafted distinct icons for various categories of POIs, ensuring that they are easily recognizable and informative (Figure 5.28).

- **Stations:** Represented by a train icon, indicating transportation hubs where responders may find critical.
- **Schools:** Symbolized by a graduation cap, these POIs highlight schools areas for evacuating individuals or where assistance may be required.
- **Hospitals:** Denoted by a the letter **H**, this POI directs responders to medical facilities for emergency care and support.
- **Churches:** Illustrated by a church icon, these locations can serve as shelters for individuals during crises or as notable locations.
- **General:** This POI is represented by a simple marker, used for various unclassified or less critical locations, allowing for flexibility in emergency planning.

FIGURE 5.28: POI Designs

These models serve not only as visual aids but also help in organizing the information presented to the users within the AR environment. The design process in Blender ensures that each POI is visually distinct, which aids first responders in quickly identifying essential locations during emergency operations.

5.6.2 POI Spawning in the AR Environment

In this subsection, we will explore the implementation of spawning Points of Interest (POIs) in the AR application. We will discuss how POIs are spawned based on messages received from the server, utilizing specific methods to ensure accurate placement and effective management within the AR environment.

Receiving POI Data from the Server

The server listens for messages from a Kafka topic (Figure 5.22), which includes instructions to spawn specific POIs (Figure 5.23). Upon receiving a message that contains the command to spawn a POI, the application processes this information using the HandleMessage function (Figure 5.24).

Spawn POI in the environment

The main function responsible for spawning POIs is SpawnPOIAtLocation (Figure 5.31), which takes a GPSCoordinate object (Figure 5.33) as an argument. When we initially recieve a spawning message from the server we call the SpawnPOIFromString method (Figure 5.29) to setup the GPS Coordinates object of the specific POI before spawning it, in order to parse this as argument on the SpawnPOIAtLocation method to spawn the POI in the AR environment. It essentially sets up the POI to

be spawned, its type and coordinates format in order to be correctly parsed to the SpawnPOIAtLocation method.

```
1
2  public void SpawnPOIFromString(string data)
3  {
4      try
5      {
6          string[] components = data.Split(',');
7          double latitude = double.Parse(components[0]);
8          double longitude = double.Parse(components[1]);
9
10         string coords = latitude + "," + longitude;
11         string poiTypeString = components[2].Substring(0, 1).ToUpper() + components[2].Substring(1).ToLower(); ;
12
13         if (!Enum.IsDefined(typeof(POIType), poiTypeString))
14         {
15             poiTypeString = "General";
16         }
17
18         POIType poiType = (POIType)Enum.Parse(typeof(POIType), poiTypeString);
19
20         GPSCoordinate coord = new() { coordinates = coords, POIType = poiType };
21
22         SpawnPOIAtLocation(coord);
23     }
24     catch (Exception ex)
25     {
26         Debug.LogError("Failed to spawn POI from string: " + ex.ToString());
27     }
28 }
```

FIGURE 5.29: Seting Up POI to spawn

The SpawnPOIAtLocation method is responsible for taking the GPS coordinates of a Point of Interest (POI) and converting them into a local Unity coordinate system, then instantiating the corresponding POI prefab at the correct location in the augmented reality environment. Here's a detailed breakdown of how the function works:

- **Calculate Object Local Position:** The method first converts the GPS coordinates (latitude and longitude) into Unity's local coordinate system using the CalculateObjectLocalPosition function. This method is covered next in more detail (Figure 5.32).
- **Instantiate Prefab:** Based on the type of the POI (e.g., school, hospital, etc.), the method retrieves the correct POI prefab from a dictionary, that has mapped the POI types with their assosiate prefabs, and then instantiates it at the computed local position (Figure 5.31).

```
1  public enum POIType
2  {
3      School,
4      Hospital,
5      Church,
6      Station,
7      General
8  }
```

FIGURE 5.30: POI Types

- **Parenting to POI Group:** For organizational purposes, the method assigns the newly created POI GameObject to a parent object named POIs. This helps keep the AR scene hierarchy clean and allows for easy manipulation of all POIs as a group if needed.
- **Setting POI Interaction Attributes:** The method then retrieves the POIInteraction component from the instantiated POI. This component is used to set various attributes of the POI, such as its name, description, and risk level. These details can be displayed when users interact with the POI in the AR environment (Figure 5.40).
- **Dictionary Management:** The POIs are stored in a dictionary called spawnedObjects, which groups POIs by their type. This organization allows for efficient management, such as enabling/disabling different types of POIs (e.g., filtering out schools) based on user preferences or application state.
- **Map and World Synchronization:** Finally, the method calls **SpawnPOIForMap** which ensures that the newly created POI is also shown in the Mapbox interface, keeping the AR environment and the map synchronized.

```
public void SpawnPOIAtLocation(GPSCoordinate coord)
{
    if (referenceCoordinatesSet)
    {
        try
        {
            Vector3 position = CalculateObjectLocalPosition(coord.Latitude, coord.Longitude);
            GameObject prefab = prefabs[coord.POIType];
            GameObject createdObj = Instantiate(prefab, new Vector3(position.x, position.y + height, position.z), Quaternion.identity);

            // Find the "POIs" GameObject and set it as the parent of the createdObj
            GameObject pois = GameObject.Find("#POIs");
            if (pois != null)
            {
                createdObj.transform.parent = pois.transform;
            }

            POIInteraction poiInteraction = createdObj.GetComponentInChildren<POIInteraction>();
            if (poiInteraction != null)
            {
                poiInteraction.poiName = coord.poiName;
                poiInteraction.poiDescription = coord.poiDescription;
                poiInteraction.riskLevel = coord.riskLevel;
            }

            if (!spawnedObjects.ContainsKey(coord.POIType))
            {
                spawnedObjects[coord.POIType] = new List<GameObject>();
            }
            spawnedObjects[coord.POIType].Add(createdObj);

            SpawnPOIForMap(coord);
        }
        catch (Exception ex)
        {
            Debug.LogError("Failed to spawn object: " + ex.ToString());
        }
    }
```

FIGURE 5.31: Method responsible for spawning POI inside the AR environment

Latitude and Longitude to Unity Coordinates

To accurately place POIs within the AR environment based on GPS coordinates, I implemented a conversion algorithm that translates latitude and longitude into Unity's local coordinate system. This method leverages the approximate distances of one degree of latitude and longitude to calculate the offset from a defined reference point.

Here is an overview what the function does:

- **Calculates the Latitude Offset:** It computes the difference in latitude between the object and the reference point, then multiplies it by an approximate value of meters per degree of latitude (111,000 meters).
- **Calculates the Longitude Offset:** It computes the difference in longitude, adjusts it based on the cosine of the latitude (to account for the Earth's curvature), and multiplies it by the same approximate meters per degree.
- **Returns a Vector3 Position:** It creates a new Vector3 object with the calculated offsets, placing the object in the Unity scene relative to the player.

```
Vector3 CalculateObjectLocalPosition(double latitude, double longitude)
{
    double latOffset = (latitude - referenceLatitude) * 111000.0;  // meters per latitude degree
    double lonOffset = (longitude - referenceLongitude) * (111000.0 * Mathf.Cos((float)(referenceLatitude * Mathf.PI / 180.0)));
    return new Vector3((float)lonOffset, 0, (float)latOffset);
}
```

FIGURE 5.32: GPS conversion to Unitys Local Coordinates

The function CalculateObjectLocalPosition (Figure 5.32) computes the position by determining the difference in latitude and longitude between the user's current location and the desired POI coordinates, which allows for the correct placement of each POI in Unity's 3D space. Although this method is not perfectly accurate due to variations in Earth's curvature, it has been tested and refined to provide satisfactory results in my application. For instance the Earth's circumference is approximately 40,075 km, and there are 360 degrees of latitude and longitude. This gives an average of about 111,319.5 meters per degree of latitude. My approach uses 111,000 meters, which is a close approximation but introduces some error. For small distances (a few kilometers), my function's approximation might be acceptable. However, over larger distances, the inaccuracies accumulate, and the object's placement might become incorrect. So in the future for even better accuracy, especially over larger distances, i will consider using the Equirectangular projection or even the Mercator projection.

GPS Coordinates Class

In the system, each POI is represented by an instance of the GPSCoordinate class, which holds critical information such as latitude, longitude, and specific details related to the POI, such as its type, name, and description (Figure 5.33).

The class is also responsible for parsing the latitude and longitude from the string format received from the server or stored in local files. This conversion allows the application to accurately position the POIs in the 3D AR environment and ensures that the geographic data is handled in a standardized format.

Here's a breakdown of the fields in the GPSCoordinate class:

- **Coordinates:** A string representation of the latitude and longitude.
- **POIType:** An enumeration that defines the category of the POI (e.g., School, Hospital, Church, etc.).
- **POI Name and Description:** Provides detailed information for users interacting with the POI.

- **Risk Level:** Additional metadata that can be used to highlight the significance or urgency of a specific POI.

```
public class GPSCoordinate
{
    [Tooltip("Input format: 'Latitude,Longitude' e.g., '34.0522,-118.2437'")]
    public string coordinates;
    public POIType POIType;
    public string poiName; // New field for the name
    public string poiDescription; // New field for description
    public float riskLevel;
    public double Latitude
    {
        get
        {
            return double.Parse(coordinates.Split(',')[0].Trim());
        }
    }

    public double Longitude
    {
        get
        {
            return double.Parse(coordinates.Split(',')[1].Trim());
        }
    }
}
```

FIGURE 5.33: GPS Coordinate Class

Predefined POIs

At the start of the application, a set of predefined POIs is spawned in the environment based on a stored list of GPSCoordinate objects. These POIs can represent important landmarks or critical points of interest, such as schools, hospitals, and stations, that are relevant to the current user's location and do not change location dynamically, they are static.

The spawning process uses the SpawnStaticPOIs function, which iterates over a list of predefined coordinates and calls the SpawnPOIAtLocation function for each. This allows the system to ensure that all relevant POIs are displayed as soon as the application starts, without needing a real-time connection to the server.

5.6.3 Visibility Issues and the POIAdjuster Solution

One of the primary challenges encountered while developing the POIs for the AR environment was maintaining the visibility of POIs over a large distance. Since POIs could be placed up to 2 kilometers away from the user, the farther POIs became significantly smaller due to distance, making them barely visible. This resulted in a lack of clarity, which impacted the user's ability to locate distant points, creating usability problems.

To address this issue, a POIAdjuster script was developed, which dynamically adjusts the size of POIs based on their distance from the user. This solution ensures that POIs remain visible and legible no matter how far they are placed within the

environment. The core of this system lies in the SizeDistance struct and the methods in the script.

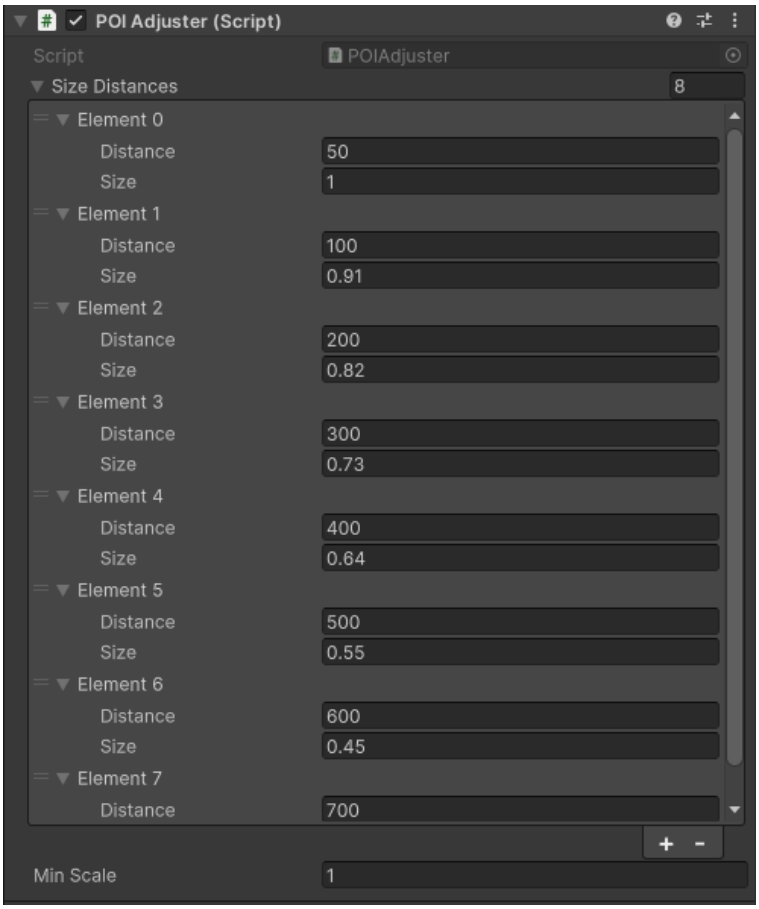


FIGURE 5.34: SizeDistance Struct

How the POIAdjuster Script Works

The POIAdjuster script is attached to each POI, and during runtime, it continually checks the distance between the POI and the user's position (Hololens 2 camera). Based on predefined distance thresholds (as configured in the inspector shown in Figure 5.34), the script calculates the appropriate scale for each POI using the method GetSizeForDistance. Here's a breakdown of the workflow:

- **Distance Calculation:** In the AdjustSizeBasedOnDistance method, the script first calculates the distance between the POI and the user's position using (Vector3.Distance).
- **Scale Calculation:** Based on this distance, it identifies the correct size from an array of size-distance pairs (Figure 5.34). It then interpolates between the two nearest size values for that specific distance, ensuring smooth scaling.
- **Scaling Adjustment:** The script adjusts the scale of the POI to the calculated size and ensures it never falls below a predefined minimum scale, maintaining legibility.

- **Drawback:** While this solution effectively keeps POIs visible at all distances, it introduces a slight reduction in immersion, as the natural decrease in size with distance is counteracted.

```
private void AdjustSizeBasedOnDistance()
{
    if (mainCamera == null || sizeDistances == null || sizeDistances.Length == 0)
        return;

    float distance = Vector3.Distance(transform.position, mainCamera.transform.position);

    // Find the appropriate size for the given distance
    float targetSize = GetSizeForDistance(distance);

    // Calculate the scale to make the POI appear the same size at different distances
    float scale = targetSize * distance;

    // Ensure the scale does not go below the minimum scale
    scale = Mathf.Max(scale, minScale);

    transform.localScale = Vector3.one * scale;
}

private float GetSizeForDistance(float distance)
{
    for (int i = 0; i < sizeDistances.Length - 1; i++)
    {
        if (distance < sizeDistances[i + 1].distance)
        {
            float t = (distance - sizeDistances[i].distance) / (sizeDistances[i + 1].distance - sizeDistances[i].distance);
            return Mathf.Lerp(sizeDistances[i].size, sizeDistances[i + 1].size, t);
        }
    }
    return sizeDistances[sizeDistances.Length - 1].size;
}
```

FIGURE 5.35: POI Adjuster Script

5.6.4 Billboarding for POI Visibility

One of the issues encountered with the POIs in the augmented reality environment is ensuring that they are always visible and oriented towards the user. Without this, POIs could end up at odd angles or out of view, making it hard for users to understand the information they convey. To address this, a **Billboard** script was implemented.

This script ensures that each POI constantly faces the user, no matter the user's position in the AR environment. The implementation works by rotating the POI on the Y-axis (horizontal plane) towards the camera, while ignoring any pitch (vertical rotation) that might misalign the object.

The script checks the position of the user (headset) and adjusts the POI's rotation accordingly in every frame through the LateUpdate() function. This guarantees that the POIs are always front-facing, improving visibility and clarity for the user.

```
void LateUpdate()
{
    if (cameraTransform == null)
    {
        cameraTransform = Camera.main.transform;
    }

    if (!isSmoothing)
    {
        // Update the object to face the camera, ignoring pitch
        Vector3 targetPosition = new Vector3(cameraTransform.position.x, transform.position.y, cameraTransform.position.z);
        targetRotation = Quaternion.LookRotation(-targetPosition + transform.position);
        transform.rotation = targetRotation;
        //Rotate parent too
        if (enableForParent && parentTranform != null)
            parentTranform.rotation = targetRotation;
    }
}
```

FIGURE 5.36: Billboarding POIs

In cases where smoother rotation is needed, a SmoothLookAtCamera() method is provided. This function allows for a more gradual adjustment, making the rotation

appear smoother over time instead of snapping immediately. The transition is handled using Quaternion.Slerp, which smoothly interpolates between two rotations, making the overall user experience more fluid.

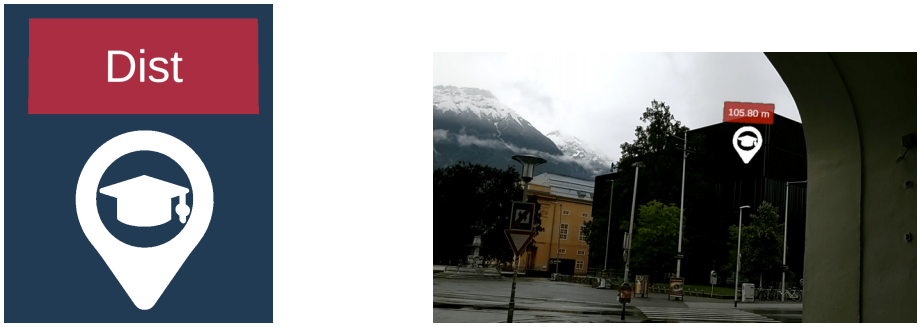
```
private IEnumerator SmoothLookAt()
{
    isSmoothing = true;
    Quaternion initialRotation = transform.rotation;
    Vector3 targetPosition = new Vector3(cameraTransform.position.x, transform.position.y, cameraTransform.position.z);
    targetRotation = Quaternion.LookRotation(-targetPosition + transform.position);

    float elapsedTime = 0f;
    while (elapsedTime < 0.5f)
    {
        transform.rotation = Quaternion.Slerp(initialRotation, targetRotation, elapsedTime * rotationSpeed);
        if (enableForParent && parentTranform != null)
            parentTranform.rotation = transform.rotation; // Rotate parent too
        elapsedTime += Time.deltaTime;
        yield return null;
    }

    transform.rotation = targetRotation;
    if (enableForParent && parentTranform != null)
        parentTranform.rotation = transform.rotation; // Rotate parent too
    isSmoothing = false;
}
```

FIGURE 5.37: SmoothLookAt() Slerp Rotation

The billboard functionality also supports adjusting both the POI itself and its parent object, which ensures that both the POI prefab and the small info panel (Figure 5.38) get rotated. This makes sure the entire structure remains correctly oriented towards the user.

5.6.5 POI Distance Helper Panel

To provide users with a better sense of spatial awareness, a distance panel is shown above each Point of Interest (POI). This panel continuously updates to display the straight-line distance between the user and the POI, helping the user understand how far away a particular location is in real-time.



(A) (B)

FIGURE 5.38: POI Distance Panel

The script responsible for this feature is part of the POIInteraction script attached to every POI. The system uses Unity's Vector3.Distance method to calculate the distance between the user's position and the POI's position in the AR environment. The distance is recalculated at regular intervals, ensuring that the displayed distance remains up-to-date as the user moves.

```
IEnumerator UpdateDistance()
{
    while (true)
    {
        Vector3 objectPositionXZ = new Vector3(transform.position.x, 0, transform.position.z);
        Vector3 userPositionXZ = new Vector3(userTransform.position.x, 0, userTransform.position.z);
        float distance = Vector3.Distance(objectPositionXZ, userPositionXZ);
        if (poiDistanceText != null)
            poiDistanceText.text = $"{distance:F2} m";
        yield return new WaitForSeconds(refresh_rate); // Wait for 1 second
    }
}
```

FIGURE 5.39: POI Distance Panel Script

A key feature of the implementation is the use of a coroutine to ensure that distance updates are smooth and efficient. By adjusting the refresh rate, we can balance performance and the need for real-time updates.

This implementation enhances user experience by providing clear and immediate distance feedback, contributing to better navigation and decision-making in the augmented reality environment.

5.6.6 Information Panel

The POIs in the AR environment are gaze-interactable, leveraging MRTK3's eye-gaze functionality. Each POI has a collider that detects when the user focuses their gaze on it. After maintaining focus for a threshold of 2 seconds, a smooth animation is triggered, which slightly rotates the POI, indicating that it is being interacted with.

Once the user has maintained gaze focus for the threshold period, an information panel appears in front of the user, displaying critical details about the POI. This information typically includes:

- Name of the POI
- Distance from the user's current location
- Risk Level, if applicable

FIGURE 5.40: POI Info Panel

The information panel automatically updates if the user gazes at another POI. The panel can also be closed by pressing on the "Close" button. This approach ensures that the interaction remains intuitive, and efficient, keeping the user informed while minimizing manual input.

```
public void ShowInfo()
{
    Vector3 objectPositionXZ = new Vector3(transform.position.x, 0, transform.position.z);
    Vector3 userPositionXZ = new Vector3(userTransform.position.x, 0, userTransform.position.z);
    float distance = Vector3.Distance(objectPositionXZ, userPositionXZ);
    string infoText = $"Name: {poiName}\nDescription: {poiDescription}\nDistance: {distance:F2} meters\nRisk Level: {riskLevel:F0}%";
    infoPanelController.ShowInfoPanel(infoText);
}
```

FIGURE 5.41: POI Info Panel Script

The ShowInfo method (Figure 5.41) handles the display logic, calculating the distance between the POI and the user, and displaying all relevant information through the information panel. The gaze interaction ensures seamless user experience in the AR environment by making the POIs dynamic, interactive, and highly informative without overloading the user with unnecessary manual interactions.

5.6.7 Manholes

Manholes are a significant safety hazard for firefighters, especially during flood-related operations. In our augmented reality system, manholes are visualized differently from other Points of Interest (POIs), emphasizing their danger. The system treats manholes as general alert areas, represented by a warning sign, rather than attempting to pinpoint their exact location due to GPS accuracy limitations, which can deviate by 5 to 10 meters.

FIGURE 5.42: Manhole in Dortmund Trials

Proximity Warning Mechanism

As the user approaches a manhole within a set radius of about 10 meters, the system triggers a proximity warning. This warning does not rely on precise GPS coordinates but encourages responders to verify the manhole's exact location manually. This ensures that first responders stay cautious in areas where manholes may pose hidden dangers beneath floodwaters.

The manhole is visualized using a custom prefab, as shown in the figures, featuring a danger symbol to emphasize its warning nature. Additionally, the alert is always visible regardless of the distance, thanks to similar size adjustment mechanisms employed for other POIs.

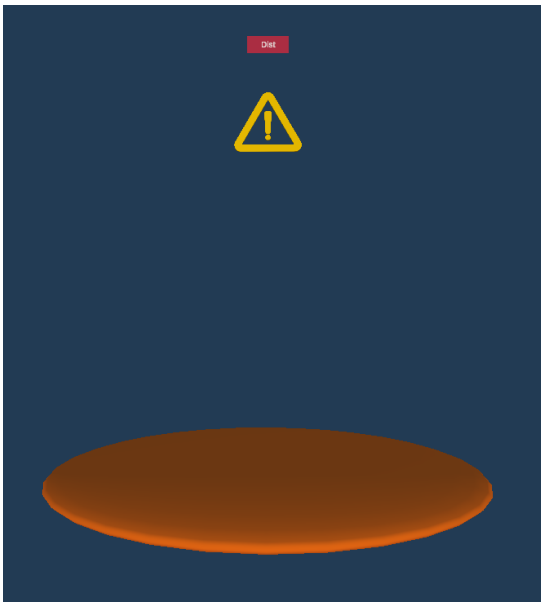
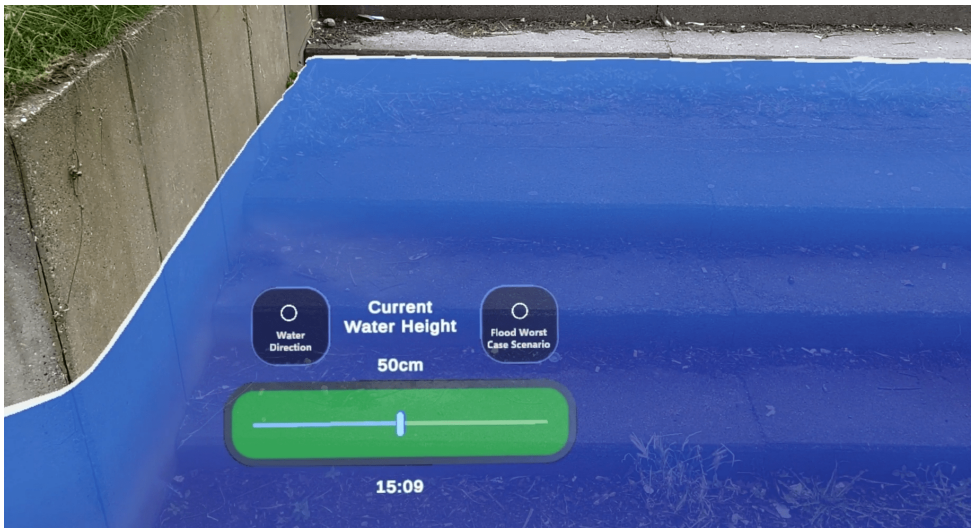


FIGURE 5.43: Manhole Design

This visual and interactive alerting system ensures that crucial hazards like manholes are highlighted effectively, helping overcome GPS limitations while still providing critical hazard awareness in potentially dangerous zones.

5.7 Water Visualization in the AR Environment

The Water Visualization feature in our augmented reality system is designed to provide first responders with a dynamic and immersive way to visualize predicted flood levels within a specified area. Utilizing real-time data, this feature allows users to interact with a 3D water plane that adapts its height based on predictions over time. This water visualization enhances situational awareness by offering a clear and engaging depiction of potential flooding scenarios, thus aiding decision-making and pre-planning in flood emergency response.



The visualization uses a custom water shader applied to a plane within the AR environment, with water heights reflecting predictions from live forecasting data. Users control the timeline of the flood forecast through a slider interface, enabling them to visualize different time frames or worst-case scenarios. This real-time interaction ensures that the information remains relevant and accurate, empowering first responders to understand the flood impact in a specific region more intuitively.

This section will explore the implementation of this flood visualization feature, detailing how the custom water shader, real-time data updates, and interactive elements like the time slider work together to create a tool for emergency preparedness and response.

5.7.1 Water Shader and Plane

In this first subsection, we will focus on describing the custom water shader and prefab that make up the water visualization feature in the augmented reality flood prediction system.

The water plane, visible in the augmented reality environment, utilizes a custom shader built for the Universal Render Pipeline (URP) in Unity, ensuring compatibility and performance on the Hololens 2 device. The water shader is designed to be lightweight yet capable of rendering a realistic water effect within the AR experience. While it may not reach the high level of realism seen in more resource-intensive simulations, it effectively provides an immersive visual representation that balances performance with visual quality.

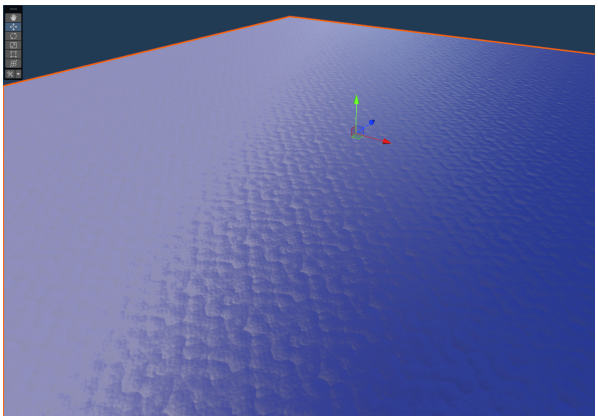


FIGURE 5.44: Water Plane Prefab

The shader applies normal maps that simulate small-scale waves on the water surface, adding an element of motion and realism to the static plane. These subtle wave effects ensure that users get the impression of flowing or dynamic water, even within the limitations of the AR environment.

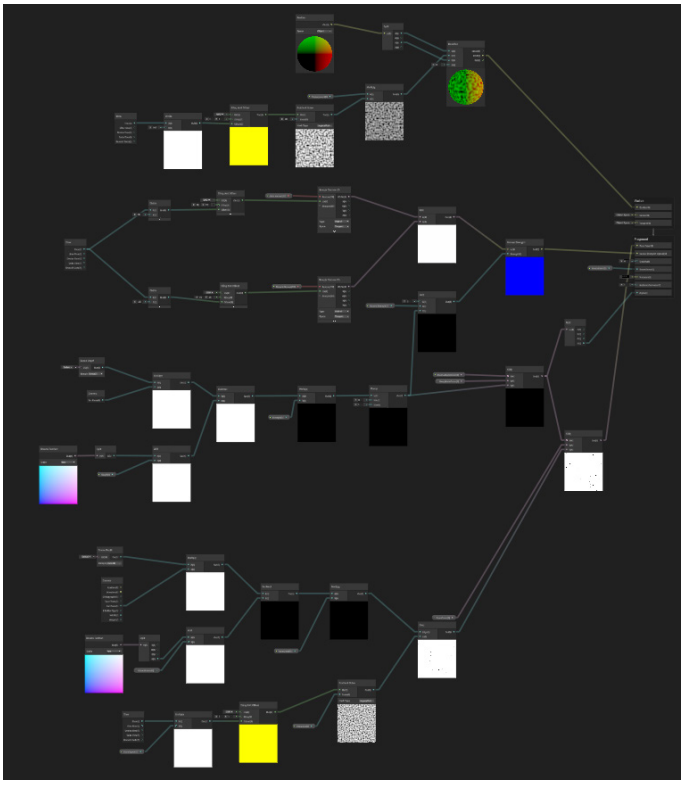


FIGURE 5.45: Water Shader Graph

Additionally, the shader is responsive to object interactions. For instance, when objects intersect or collide with the water plane, a foam effect is triggered, represented by white highlights around the point of impact. This visual feedback helps users better understand the depth and movement of the water within their physical space.

The foam further enhances the realism by depicting how water reacts to nearby objects, ensuring that responders have a more intuitive sense of the surrounding environment during a flood emergency.

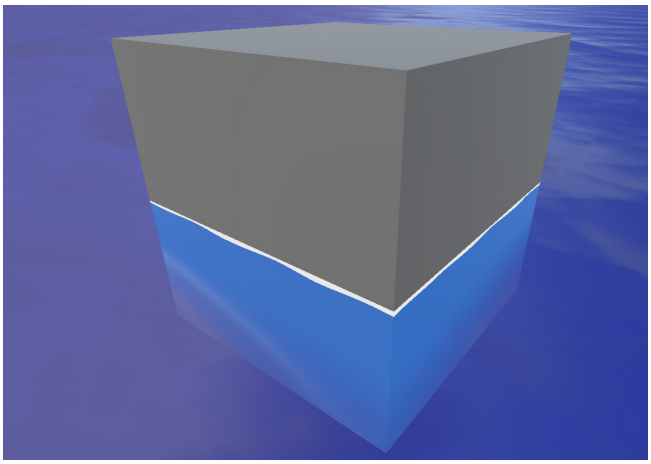


FIGURE 5.46: Water Collision Indicator

The shader's nature and functionality, combined with the AR environment's need for efficient rendering, make it an effective solution for visualizing predicted flood levels in real-time, offering a balance between aesthetics and performance.

5.7.2 Spatial Mapping for Immersive Water Visualization

To enhance the realism and immersion of the flood prediction application for firefighters and first responders, spatial mapping of the environment is essential. By utilizing the spatial awareness capabilities of the HoloLens 2, the application can generate meshes of the surrounding area, allowing the virtual water plane to interact seamlessly with real-world surfaces. This interaction enables the water to collide with and be obstructed by physical objects, providing users with a more accurate and intuitive visualization of predicted water heights in a flood scenario.

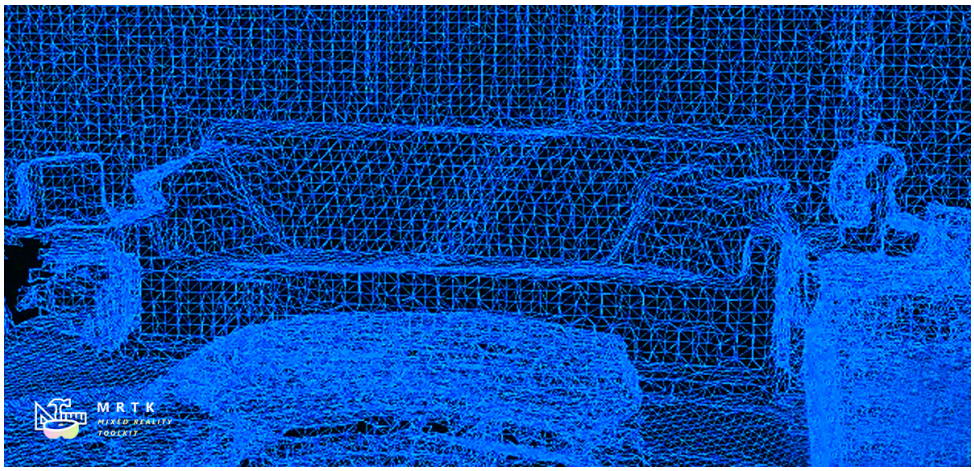


FIGURE 5.47: MRTK Spatial Awareness

Implementation Using ARMeshManager

The spatial mapping functionality is implemented using the ARMeshManager component from the Mixed Reality Toolkit 3 (MRTK3). The ARMeshManager is responsible for generating and updating mesh data from the environment in real-time. It captures spatial features by scanning the surroundings and creates a mesh representation that the Unity physics engine can use for collision detection.

The primary roles of the ARMeshManager

- **Mesh Generation:** Continuously scanning the environment to create a detailed mesh that represents physical surfaces.
- **Collision Integration:** Allowing the water plane to interact with the generated meshes, enabling realistic water behavior and immersion.
- **Mesh Updating:** Refreshing the mesh data at specified intervals to accommodate changes in the environment or user movement.

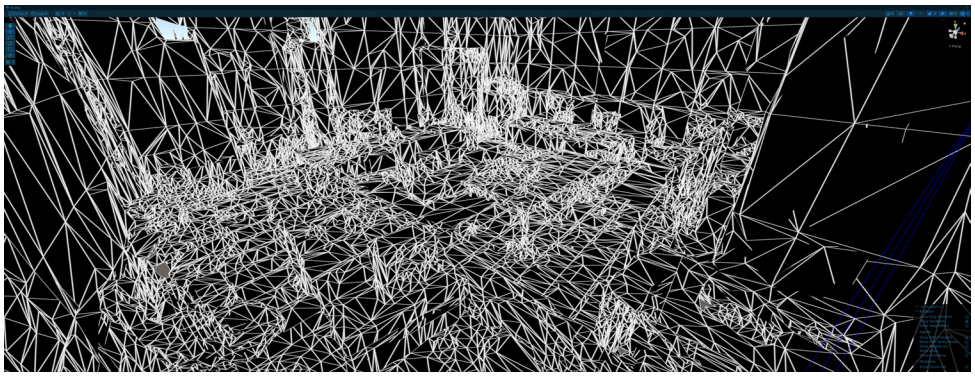


FIGURE 5.48: Scanned Lab Room

Performance Optimization

While the ARMeshManager provides critical functionality for spatial mapping, it introduces significant computational load. Real-time mesh generation and updating are resource-intensive processes, leading to a noticeable drop in frame rate on the HoloLens 2 device. To limit this challenge, several optimizations were implemented:

- **Reduced Update Frequency:** The mesh updating process was modified to occur at longer time intervals rather than every frame. This reduces the computational load by decreasing the frequency of heavy processing tasks.
- **Mesh Cleanup on Feature Disablement:** Adjustments were made to clear unnecessary mesh data when certain features are disabled. This frees up memory and processing resources.
- **Selective Scanning:** Limiting the scanning area or focusing on critical regions reduces the amount of data processed, further enhancing performance.

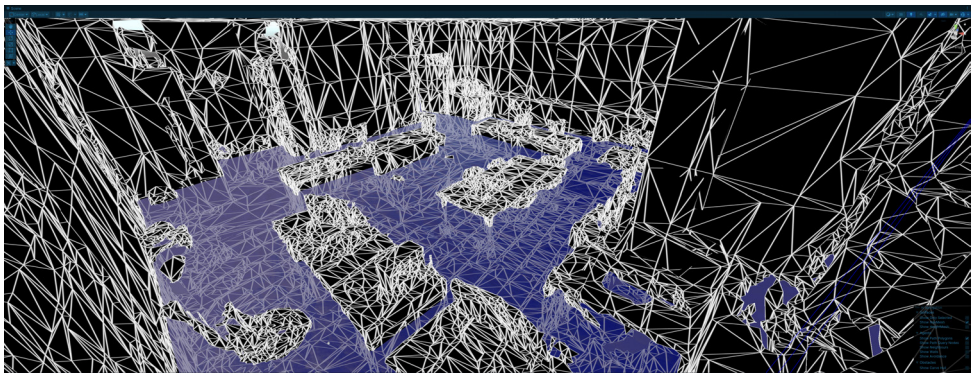


FIGURE 5.49: Scanned Lab Room with Water Visualization enabled

These optimizations resulted in a more stable application performance, limiting frame rate drops and preventing crashes. Although there is still some reduction in frame rate compared to scenarios without spatial mapping, the trade-off is acceptable given the enhanced immersion and functionality.

User Experience Enhancement

By enabling the water plane to collide with and be obstructed by real-world objects, users gain a more intuitive understanding of potential flood impacts and predicted heights in their immediate environment. This immersive experience is crucial for emergency planning and response, as it allows first responders to visualize flood scenarios accurately.

5.7.3 Enabling Water Visualization and Accurate Water Placement

The water visualization feature is enabled through a toggle button within the hand menu. Upon activation, users are prompted to look at the ground, ensuring the system scans the surrounding environment to accurately position the water plane. This section outlines the process behind enabling the feature and the logic used for positioning the water plane.

Water Plane placement and Scanning

The process of enabling the water feature starts a coroutine in the WaterLevelController script, which activates the ARMeshManager for spatial awareness and also activates the panel to prompt the user to look at the ground.

```
private IEnumerator StartScanningAndSpawnWater()
{
    arMeshManager.SetActive(true); // Enable scanning
    instructionPannel.SetActive(true); // Show the instruction panel
    yield return new WaitForSeconds(scanDuration); // Wait for the scanning duration
    instructionPannel.SetActive(false); // Hide the instruction panel
    waterPlaneScript.SetWaterVisibility(_isWaterVisible);
    waterPlaneScript.BringWater(); // Spawn the water after scanning
    // Enable slider gameobject
    sliderGameObject.SetActive(true);
}
```

FIGURE 5.50: Water Start Coroutine

The ARMeshManager scans the surrounding environment, creating meshes that the water plane will interact with. The coroutine waits for a specified duration (5 seconds) for the environment to be scanned correctly, ensuring accurate mesh generation for proper collision detection. This ensures that the ground meshes will be generated so that the calculation and identification of the ground's Y position can be done correctly.

The height calculation process contains the CalculateGroundHeight() function, which performs a downward raycast from the user's camera, detecting the Y-coordinate of the ground. (Figure 5.51).

```
public float CalculateGroundHeight()
{
    // Create a ray pointing downward from the camera's position
    Ray ray = new Ray(mainCamera.transform.position, Vector3.down);
    RaycastHit hit;

    // Perform the raycast
    if (Physics.Raycast(ray, out hit, rayDistance, groundLayer))
    {
        _groundHeight = hit.point.y;
        Debug.Log($"Collision Point: {hit.point}");
        return _groundHeight;
    }
    else
    {
        Debug.Log("No ground detected within the ray distance.");
        return -1112f; // Return a negative value to indicate no ground detected
    }
}
```

FIGURE 5.51: Reference Ground Y Calculation

Once the ground height is calculated, the water plane is positioned at the detected position plus a value representing the predicted flood level which is provided by the server. The BringWater() function in the WaterPlane script is responsible for this adjustment, ensuring that the water plane's height is dynamically updated based on user interaction and predicted flood levels.

The water plane is placed based on the predicted water height for the selected time interval, enabling first responders to visualize the flood scenario effectively.

```
public void BringWater()
{
    float groundHeight = heightCalculator.CalculateGroundHeight();
    if (groundHeight != -1112f)
    {
        // Position the water plane based on the calculated ground height and slider value
        float waterHeight = waterHeights[Mathf.Clamp((int)WaterLevelController.CurrentSliderValue - 1, 0, waterHeights.Length - 1)];
        float adjustedHeight = groundHeight + waterHeight;
        StartCoroutine(SmoothMove(new Vector3(heightCalculator.mainCamera.transform.position.x, adjustedHeight, heightCalculator.mainCamera.transform.position.z)));
    }
}
```

FIGURE 5.52: Water Placement Method

5.7.4 Prediction Slider and Water Height Updates

The prediction slider is a crucial interactive component in the AR water visualization system, allowing users to visualize flood predictions over time. This UI slider, adapted from the MRTK3 prefab, enables users to select between three predicted future time points, with each interval set 10 minutes apart. For example, if the initial time is 2:00, the three prediction points would represent flood conditions at 2:10, 2:20, and 2:30.

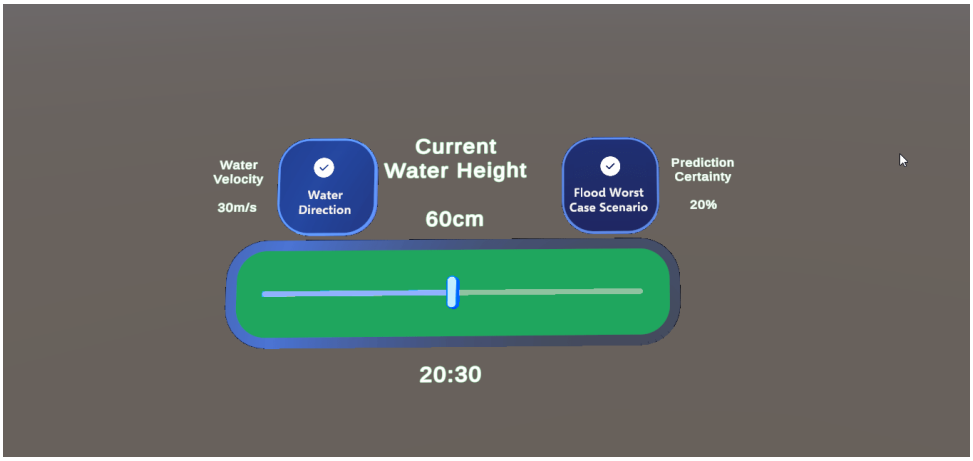


FIGURE 5.53: Prediction Slider

Slider Features

- **Real-Time Prediction:** The slider displays flood conditions such as current water height, water velocity, and prediction certainty. As the user adjusts the slider, these values update in real time to reflect the selected prediction interval.
- **UI Design:** The design is intuitive and easy to use, showing the current water height in centimeters, the water velocity in meters per second, and the worst-case flood scenario with its prediction certainty, all of which help first responders make informed decisions.
- **Additional Features:** The slider also includes additional features that enhance information like Water direction visualization and Worst case scenario buttons.

Prediction Logic

When the user interacts with the slider, the system sends the current time and GPS location to the server, which processes this information and retrieves the relevant flood predictions from its database. The server responds with predictions for three different time points, each spaced 10 minutes apart, and these values are stored in Unity. The water height is updated accordingly as the slider moves, providing a dynamic and immersive experience for users visualizing the flood impact in the specified area.

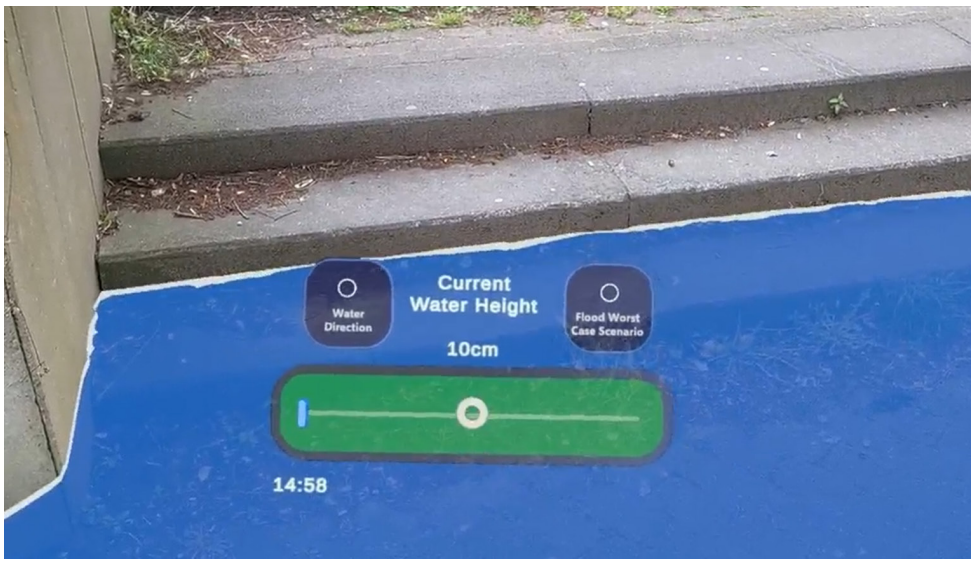


FIGURE 5.54: Selecting Prediction Time

Server-Side Processing

The prediction values are processed through the server using a table of flood data that matches the GPS location with the requested time intervals. For instance, the server might retrieve prediction data for the times 2:02, 2:12, and 2:22. The data is then returned to the client, and the slider updates in Unity according to these predictions.

```
function getValuesForTime(inputTime, dataMap) {
  // Convert the inputTime to minutes
  const [inputHours, inputMinutes] = inputTime.split(":").map(Number);
  const inputTotalMinutes = inputHours * 60 + inputMinutes;

  let value1 = null,
    value2 = null,
    value3 = null;

  // Ensure dataMap is properly formatted
  if (!(dataMap instanceof Map)) {
    throw new Error("dataMap is not a Map instance");
  }

  for (const [key, value] of dataMap) {
    // Ensure key is properly formatted
    if (typeof key !== "string" || !key.includes(", ")) {
      console.error(`Invalid key format: ${key}`);
      continue;
    }

    const [date, time] = key.split(", ");

    // Ensure time is properly formatted
    if (typeof time !== "string" || time.split(":").length !== 3) {
      console.error(`Invalid time format: ${time}`);
      continue;
    }

    const [hours, minutes] = time.split(":").map(Number);
    const totalMinutes = hours * 60 + minutes;

    if (totalMinutes === inputTotalMinutes) value1 = value;
    if (totalMinutes === inputTotalMinutes + 10) value2 = value;
    if (totalMinutes === inputTotalMinutes + 20) value3 = value;
  }

  return `${value1 ?? "null"},${value2 ?? "null"},${value3 ?? "null"}`;
}
```

FIGURE 5.55: Server Prediction Heights Retrieval

This interactive feature ensures that first responders can quickly visualize and comprehend the flood's progression, making the slider a key tool for real-time flood prediction interaction.

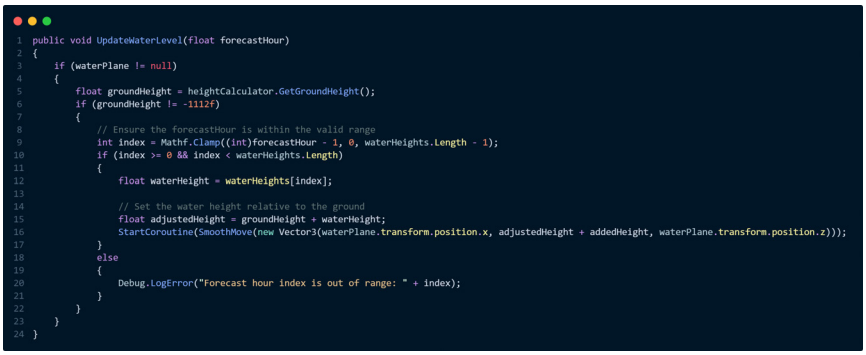
Water Height Update

The water height updating process begins when the Unity application retrieves prediction data from the server. The server sends back three distinct values for predicted water heights based on the user's GPS location and selected time interval. Each prediction corresponds to a different future point in time (e.g., 10, 20, or 30 minutes into the future). Upon receiving this message, the application parses the data to store these predicted heights in an array.

```
if (message.Contains("water"))
{
    Debug.Log("Water level: " + message);
    string[] parts = message.Split(',');
    float[] waterHeights = new float[3];
    // Parse the first 3 parts of the message as water heights
    // Parse the first three values to floats
    for (int i = 0; i < 3; i++)
    {
        if (float.TryParse(parts[i], System.Globalization.NumberStyles.Float, System.Globalization.CultureInfo.InvariantCulture, out float result))
        {
            waterHeights[i] = result / 100;
        }
        else
        {
            Debug.LogError($"Failed to parse '{parts[i]}' as float.");
            waterHeights[i] = 0f; // Assign a default value or handle error as needed
        }
    }
    Debug.Log("Water heights: " + string.Join(", ", waterHeights));
    waterController.UpdateWaterHeights(waterHeights);
}
```

FIGURE 5.56: Application Prediction Data Handling

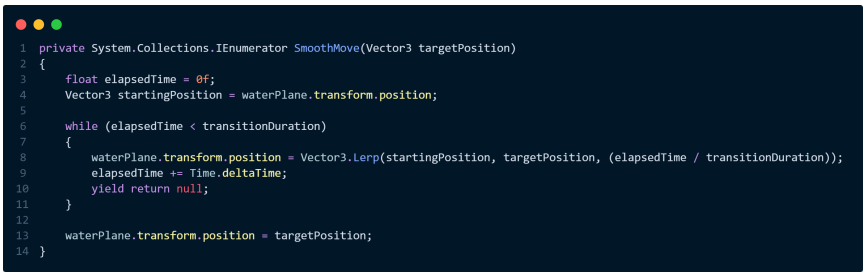
Next, the UpdateWaterLevel() method (Figure 5.57) is called. This method is responsible for handling the visualization of the water height in Unity's 3D environment. It first checks if the ground height has been properly calculated using the heightCalculator.GetGroundHeight() function, which is essential for ensuring that the water plane is positioned correctly in relation to the terrain. The predicted water height is then retrieved from the stored values and combined with the ground height to adjust the water plane's position in the scene. The result is a smooth adjustment of the water level, which visually represents the flood predictions at the selected future time.



```
public void UpdateWaterLevel(float forecastHour)
{
    if (waterPlane != null)
    {
        float groundHeight = heightCalculator.GetGroundHeight();
        if (groundHeight != -1112f)
        {
            // Ensure the forecastHour is within the valid range
            int index = Mathf.Clamp((int)forecastHour - 1, 0, waterHeights.Length - 1);
            if (index >= 0 && index < waterHeights.Length)
            {
                float waterHeight = waterHeights[index];

                // Set the water height relative to the ground
                float adjustedHeight = groundHeight + waterHeight;
                StartCoroutine(SmoothMove(new Vector3(waterPlane.transform.position.x, adjustedHeight + addedHeight, waterPlane.transform.position.z)));
            }
            else
            {
                Debug.LogError("Forecast hour index is out of range: " + index);
            }
        }
    }
}
```

FIGURE 5.57: Updating Water Level Method

The smooth transition of the water plane is achieved through the SmoothMove() function, which allows for gradual, fluid movement of the water between different height values. This ensures that the visualization is not abrupt, enhancing the realism and user experience in the AR environment.



```
private System.Collections.IEnumerator SmoothMove(Vector3 targetPosition)
{
    float elapsedTime = 0f;
    Vector3 startingPosition = waterPlane.transform.position;

    while (elapsedTime < transitionDuration)
    {
        waterPlane.transform.position = Vector3.Lerp(startingPosition, targetPosition, (elapsedTime / transitionDuration));
        elapsedTime += Time.deltaTime;
        yield return null;
    }

    waterPlane.transform.position = targetPosition;
}
```

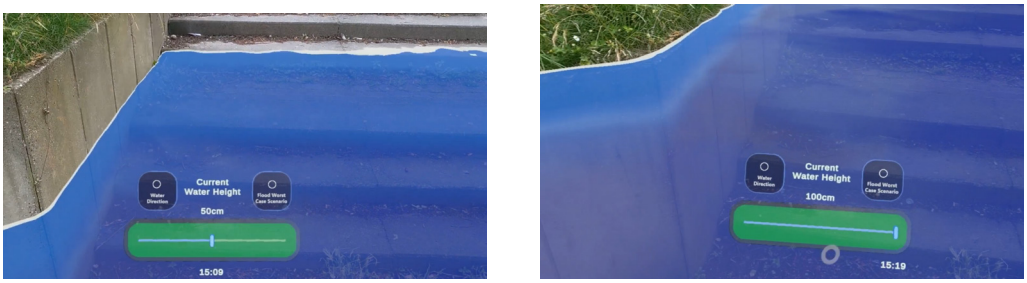
FIGURE 5.58: Smooth Water Transition

In terms of user interaction, the slider mechanism plays a central role. When a user adjusts the time slider, an event (OnSliderValueChanged()) is triggered (Figure 5.59). This event recalculates the water height based on the selected time, calls the UpdateWaterLevel() (Figure 5.57) method, and updates the information panel. This panel displays the current water height, velocity, and prediction certainty, giving users a real-time understanding of the flood risk at different points in time. Although the certainty and velocity values are experimental at this stage, they provide a foundation for future data integration and enhanced prediction accuracy.

```
private void OnSliderValueChanged(SliderEventData eventData)
{
    CurrentSliderValue = eventData.NewValue;
    if (waterPlaneScript != null)
    {
        waterPlaneScript.UpdateWaterLevel(CurrentSliderValue);
        ChangePOIColorRandom();
        UpdateWaterLevelInfo();
        PredictedTime.text = PredictionTime(CurrentSliderValue);
    }

    worstCaseHeightText.text = "+ " + waterPlaneScript.worstCaseWaterPlaneHeight * 100 + "cm";
    //FOR TESTING PURPOSES
    //Make uncertainty text random between 90-100% on slider position 1, between 60-90% on slider position 2 and between 20-60% on slider position 3
    float randomValue = Random.Range(0.0f, 1.0f);
    if (CurrentSliderValue == 1)
    {
        UncertaintyText.text = $"Prediction Certainty\n\n{randomValue * 10 + 90:F0}%";
        velocityText.text = "Water Velocity\n\n0.2 m/s";
    }
    else if (CurrentSliderValue == 2)
    {
        UncertaintyText.text = $"Prediction Certainty\n\n{randomValue * 30 + 60:F0}%";
        velocityText.text = "Water Velocity\n\n2 m/s";
    }
    else
    {
        UncertaintyText.text = $"Prediction Certainty\n\n{randomValue * 40 + 20:F0}%";
        velocityText.text = "Water Velocity\n\n3 m/s";
    }
    ////////////////////////////////////

}
```

FIGURE 5.59: Slider Interaction Event

The overall process of updating the water height based on prediction data and slider interaction results in a dynamic, user-friendly and interactive flood visualization tool for first responders. Here is an example of before and after the water height update for prediction values of 50cm and 1meter respectively.



(A) Water visualization at 50 cm (B) Water visualization at 1 meter

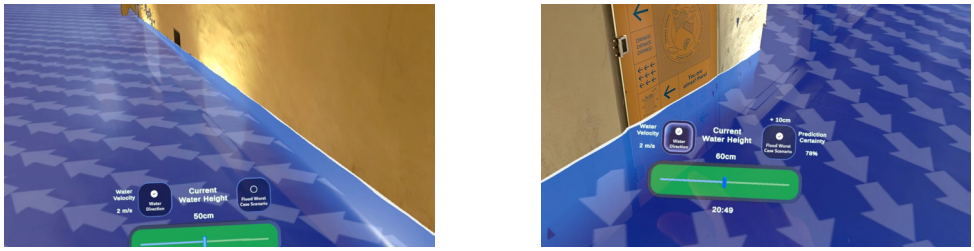
FIGURE 5.60: Water Visualization Transition from 50 cm to 1 meter

5.7.5 Additional Features

In addition to the basic water visualization, the AR system includes two additional features that offer more detailed insights into flood predictions. Water Direction and Worst Case Scenario. Both features are toggles within the slider panel, enabling users to access these functionalities as needed.

Water Direction Visualization

When this feature is enabled, the water plane is overlaid with directional arrows that indicate the predicted flow of water. These arrows serve as visual cues to help responders understand the flow patterns and direction of floodwaters at any given time. The arrows themselves are created using a Unity shader, allowing for dynamic adjustments in both rotation and speed to reflect water direction and flow intensity.



(A) (B)

FIGURE 5.61: Water Direction Example

The direction of these arrows is determined by experimental data from the server, making the visualization both context-specific and dynamic. This feature is particularly useful in scenarios where responders need to know not only the water levels but also where the water is likely to flow, helping in risk mitigation and rescue planning.

Worst Case Scenario Visualization

The Worst Case Scenario feature offers users an understanding of the maximum potential flood height based on predictions. When activated, this feature adjusts the water plane's height to show the worst-case predicted scenario for flooding.

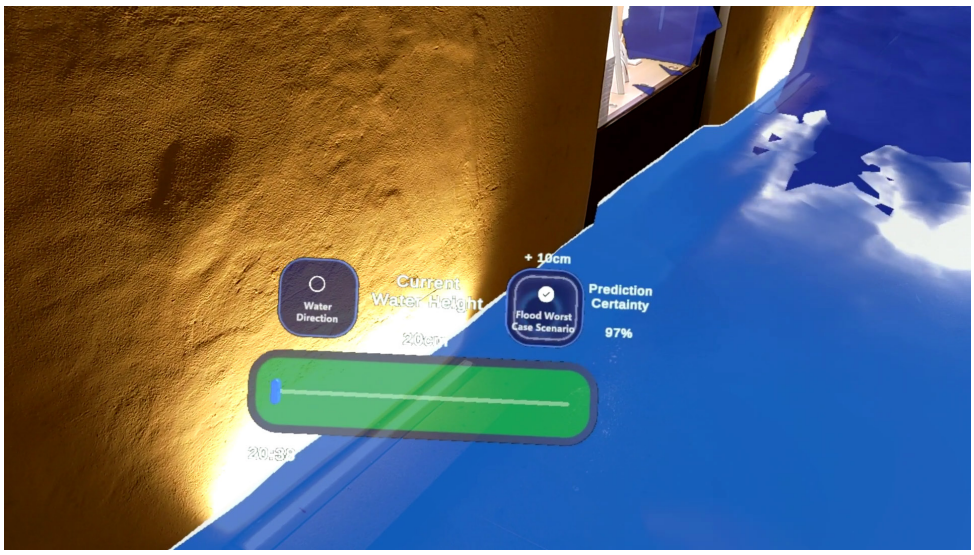


FIGURE 5.62: Worst Case Scenario Feature

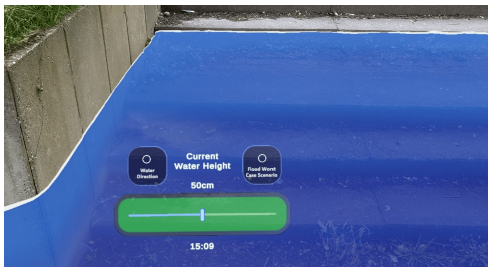
Along with displaying the regular prediction, the system adds a predicted extra height in the visualization, which accounts for the worst-case outcome. This is

shown in the slider panel, providing an overlay of both the regular and worst-case water heights. The certainty of the prediction, although experimental, is also displayed, giving users an idea of the confidence level in the forecast.

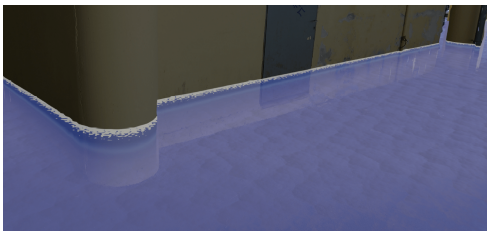
These additional features enhance the water visualization system, offering more granular insights into flood patterns and risks.

5.7.6 Water Visualization Examples and Figures

This subsection provides a visual representation of the water plane in action within the AR application. The following images show how the water visualization integrates into the AR environment, creating an immersive experience for users, particularly first responders.

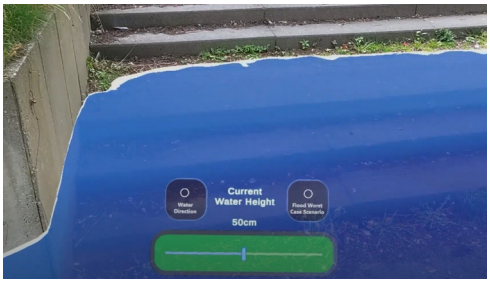


(A)



(B)

(A)



(B)

5.7.7 Unity Hierarchy

The organization of the Unity hierarchy is crucial for maintaining a clean and efficient project structure. This sub-section provides the hierarchy for al the features included in this chapter and how they are organized within the Unity Editor.

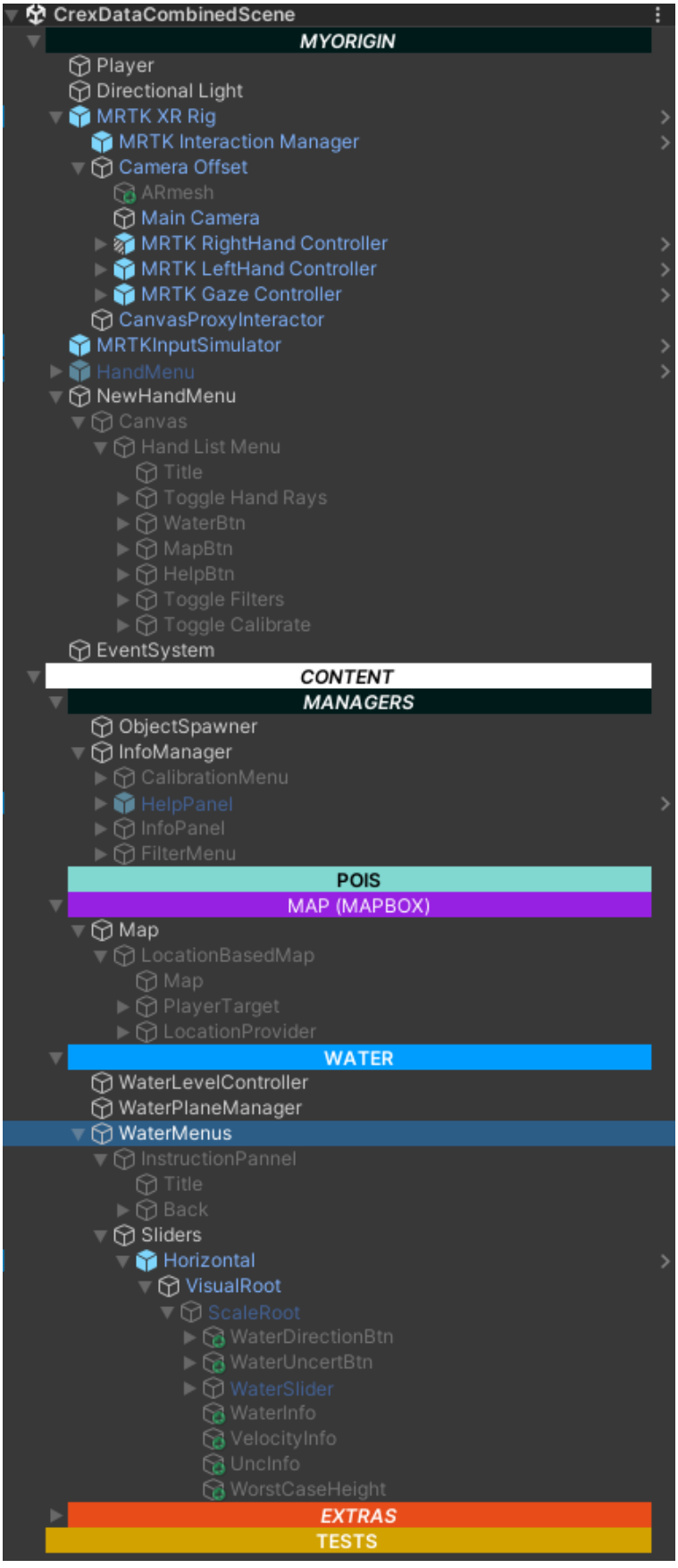


FIGURE 5.65: Unity Editor Hierarchy

5.8 Conclusion

By utilizing advanced features such as POI interaction, water visualization, spatial awareness, and dynamic flood prediction updates, the system delivers a comprehensive representation of potential flood hazards. Each element, from real-time data integration to user-friendly interface design, contributes to a cohesive system aimed at improving situational awareness in high-risk environments. This chapter has outlined the key technical components and features, establishing the groundwork for how the system operates and its capabilities in real-world applications.

Chapter 6

Evaluation

6.1 Introduction

The evaluation of the AR system prototype was done in relation to the CREXDATA project, aimed to gather feedback on its usability and functionality in real-world emergency response scenarios. These trials were crucial in assessing the system's potential to enhance decision-making processes and improve the safety and efficiency of first responders during flooding emergencies. The trials were conducted in two key locations, Innsbruck and Dortmund, each contributing valuable insights into the system's design and its application in different operational settings.

FIGURE 6.1: Trials Team

6.2 Locations and Overview of Trials

The trials were conducted in two key locations, Innsbruck and Dortmund, each contributing valuable insights into the system's design and its functionality in different operational settings. The trials involved participants from various backgrounds, including fire brigade experts, emergency responders, and researchers, who tested the AR system in simulated flooding scenarios. All of the participants were introduced to the AR system and its functionalities and also provided with guidance on how to use the system and the Hololens 2 device. The trials were conducted in a controlled environment, where participants were asked to perform specific tasks using the AR system, such as map visualization, POI display, water level prediction, and real-time information integration. The trials were recorded and analyzed to identify issues, areas for improvement and future requests to make the system more usable and effective in real-world emergency response scenarios.

(A) (B)

FIGURE 6.2: Trials from both cities

6.2.1 Innsbruck

The first trial took place in Innsbruck with a focus group of fire brigade experts and emergency responders. During this trial, participants were introduced to the AR system, where they tested its various functionalities, such as map visualization, POI display, water level prediction, and real-time information integration.

(A)

(B)

(C)

(D)

FIGURE 6.3: Trials in city of Innsbruck

Observations and first feedback

Participants noted that the displayed map in the AR interface was initially positioned in their immediate field of vision, which obstructed their view of the environment. Based on this feedback, the map's location was adjusted to the lower edge of the field of vision, improving safety and spatial awareness. Similarly, difficulties were encountered with the main UI, which was initially centered in front of the user's view. To resolve this issue, the UI was repositioned onto the user's palm (Figure 5.4)for easier interaction, leading to positive feedback regarding this adjustment.

6.2.2 Dortmund

Following the Innsbruck trial, the AR system underwent several optimizations, including the UI adjustments and the repositioning of the map and menu elements. The revised system was tested in Dortmund with a new group of firefighters, who found the interface to be leaner and more intuitive. The adjustments made based on the Innsbruck feedback were well-received, and additional comments emphasized the innovative nature of the water level visualization. Responders appreciated the system's ability to display real-time water flow direction, which could help them assess hazards such as drifting objects in floodwaters.

(A) (B)

FIGURE 6.4: Trials in city of Dortmund

6.3 Feedback and Observations from the Trials

The firefighters provided detailed feedback during both trials, focusing on usability and the relevance of the displayed information for emergency missions. The water level visualization was particularly praised, with many experts noting its usefulness for predicting the potential impact of rising water levels on the surrounding area. However, they suggested improvements, such as reducing the constant visibility of the water prediction slider, which was seen as distracting during certain tasks. This was addressed by replacing the slider with a smaller, less intrusive button.

Feedback on AR Functionalities

The AR functionalities were rated across various criteria, as seen in the boxplot evaluations. The tool was particularly well-regarded for its ability to provide mission-relevant information about critical points of interest (POIs) and vulnerable elements in the environment.

- **Illustration of Vulnerable Elements (POI-related):** The tool's ability to visualize vulnerable elements, such as manholes and critical infrastructure, was rated highly, with a score of 4.0. This feature was seen as essential for informing decisions about rescue operations in hazardous areas.
- **Manhole Visualization:** As an integral feature for ensuring firefighter safety, the representation of manholes as general alert areas received positive feedback. Firefighters appreciated the warning system that notified them of potential dangers, especially in flood scenarios.
- **Routing and Orientation:** The general routing ability that the system gave the user was rated positively, with experts noting that it provided them with valuable information to plan safe evacuation routes or deployment strategies during emergency situations. The illustration of planning for possible evacuation routes and routing forces was scored favorably, though with a slightly lower rating (around 3.65) compared to other features.

6.3.1 Visual Data on AR System Usability

The evaluation also involved a detailed analysis of the usability data collected from the trials. The boxplot visualizations provide insight into how the AR functionalities were rated across specific criteria, such as POI-related features and routing/orientation functionalities. These visual data points underscore the overall positive reception of the AR system while also indicating areas for further refinement.

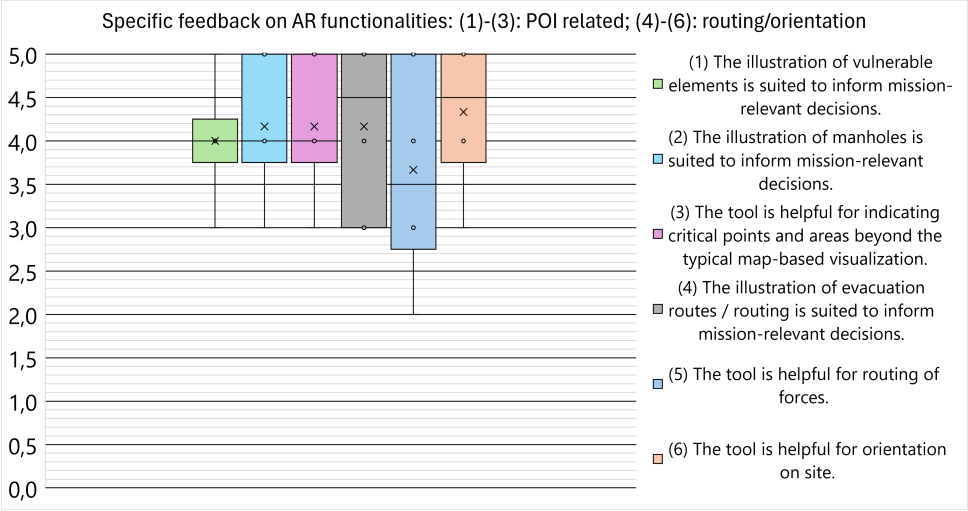


FIGURE 6.5: Feedback analysis on AR Functionalities

6.3.2 Challenges

In addition to the positive feedback, several challenges were highlighted during the trials. Participants noted that using the AR system placed additional mental demand on the responders, as it required them to balance attention between the AR interface and their physical surroundings. Moreover, the current AR devices were found to have limitations, such as a lack of compatibility with protective equipment and the waterproof limitation, which is crucial for emergency responders working in adverse conditions. Another significant challenge was the reliance on a stable internet connection for data transfer, which might not always be available in remote or crisis situations, potentially impacting the system's reliability in the field. Addressing these challenges will be essential for future iterations of the AR system to ensure it can perform effectively in a broader range of emergency scenarios.

6.3.3 Feedback Summary

The AR system received positive feedback overall, with users logging suggestions for future development. Several responders highlighted the potential for AR to support more advanced decision-making processes, particularly in scenarios where the location of manholes and other critical infrastructure could be visualized in real-time. The responders also emphasized the importance of ensuring that the AR system could function seamlessly with other emergency response technologies, including traditional communication methods and protective gear.

Despite the challenges, the participants from both Innsbruck and Dortmund agreed that the AR system showed great potential for improving operational efficiency during flooding and other emergency scenarios.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis introduced and detailed the development of an Augmented Reality (AR) system aimed at enhancing situational awareness and decision-making for emergency responders during flood scenarios. Through real-time 3D visualizations of water levels, map and critical Points of Interest (POIs), the system promises to allow first responders to better anticipate hazards and navigate complex environments under challenging conditions. This was achieved by integrating spatial data with AR capabilities, providing users with immersive, real-time flood predictions and relevant warnings.

The system underwent trials in two European cities, Innsbruck and Dortmund, where professional firefighters provided feedback on its performance. The trials highlighted the potential of the AR system to revolutionize on-site disaster response by offering real-time insights that could inform mission-critical decisions. Feedback from the participants was largely positive, particularly regarding the intuitive nature of the system's water visualization and the clear representation of potential risks associated with POIs like manholes and vulnerable areas.

However, certain limitations were identified, such as the additional mental demand required to operate the system and hardware constraints, including the non-waterproof nature of the AR devices and the high costs involved. These factors present areas of future improvement to further refine the system for real-world deployment.

7.2 Future Work

Looking ahead, several improvements and features are planned to enhance the AR system based on the feedback received during the trials.

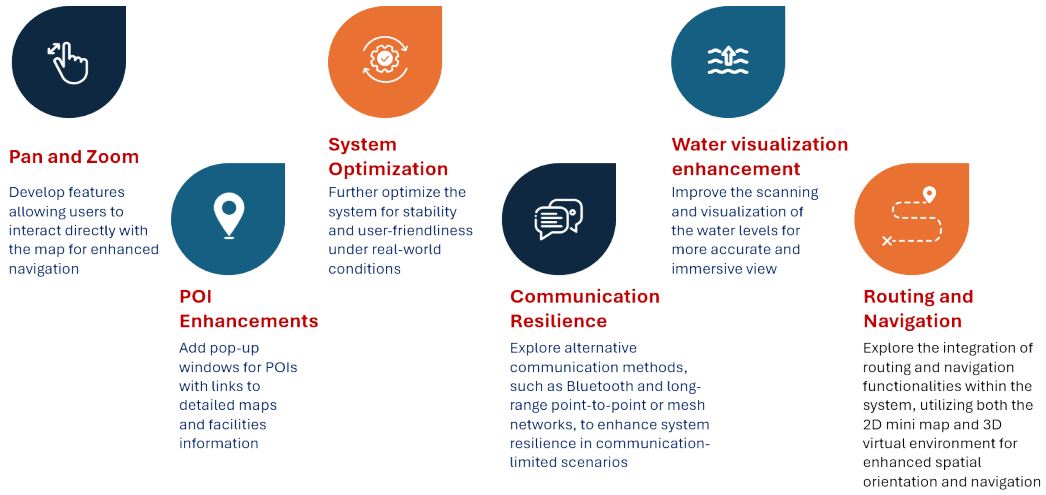


FIGURE 7.1: Future Work Outline

First, incorporating a pan and zoom functionality will enable users to interact more intuitively with the map interface, enhancing control over navigation and mission planning. This will allow first responders to zoom in on critical areas for more detailed analysis while maintaining an overview of the situation.

In terms of system optimization, ongoing efforts will focus on refining the system's stability and performance under real-world conditions. This involves addressing the computational load of the AR device, particularly with spatial awareness features like mesh generation and also structuring the application in a way so that certain information and visualization can be viewed at each time and not overload the user with too much information. Improved optimization and refactor will help ensure that the system remains responsive and easy to use in high-pressure situations.

Enhancements in water visualization are also a priority, aimed at making flood predictions even more accurate and immersive. These improvements will focus on better rendering of water movement and collision detection, creating a more realistic flood visualization for users. This is essential to provide emergency responders with the most precise visual representation of flood predicions.

The system's POI functionality will be expanded by integrating features such as pop-up windows linked to detailed maps and facility information. This will provide users with more in-depth context regarding critical POIs, enabling more informed decision-making during emergencies. Additionally, introducing the ability to manually rank POIs by risk score will give responders a visual cue to prioritize actions based on the severity of the risks posed by different elements.

Given the potential for communication limitations during emergency situations, future work will explore alternative methods such as Bluetooth, mesh networks, and long-range point-to-point communication. This will improve the system's resilience and ensure that it remains operational.

Lastly, the routing and navigation features will be enhanced to add a guidance and offer a more comprehensive view of safe evacuation routes and critical areas. By combining 2D mini-maps with a 3D virtual environment routing, the system will give users a better guidance and improved orientation and locomotion on in-site missions for better pre-planning. This will support safe movement through hazardous areas and assist in coordinating response efforts effectively.

With these planned enhancements, the AR system will become an even more valuable tool for emergency responders. The continuous refinement of these functionalities, along with the integration of user feedback, will ensure that the system evolves into a robust and reliable tool for managing flood emergencies.

Bibliography

Andrade, Gustavo Vargas de et al. (2022). "Towards an augmented reality application to support civil defense in visualizing the susceptibility of flooding risk in Brazilian urban areas". In: *International conference on computational science and its applications*. Springer, pp. 494–506.

Barmpas Zachariadis, Nikolaos (2020). *Development of an iOS, Augmented Reality for disaster management*. eng. Student Paper.

Campos, Alexandre et al. (2019). "Mobile augmented reality techniques for emergency response". In: *Proceedings of the 16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pp. 31–39.

Chalimas, Theodoros and Katerina Mania (2023). "Cross-Device Augmented Reality Systems for Fire and Rescue based on Thermal Imaging and Live Tracking". In: *2023 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pp. 50–54. DOI: 10.1109/ISMAR-Adjunct60411.2023.00018.

Daskalogrigorakis, Grigoris et al. (Oct. 2022). "Glance-Box: Multi-LOD Glanceable Interfaces for Machine Shop Guidance in Augmented Reality using Blink and Hand Interaction". In: pp. 315–321. DOI: 10.1109/ISMAR-Adjunct57072.2022.00070.

Erra, Ugo et al. (Mar. 2018). "Mobile Augmented Reality For Flood Events Management". In: *International Journal of Sustainable Development and Planning* 13, pp. 418–424. DOI: 10.2495/SDP-V13-N3-418-424.

Gomez, F. J. et al. (2024). "Probabilistic Flood Inundation Mapping through Copula Bayesian Multi-Modelling of Precipitation Products". In: *Natural Hazards and Earth System Sciences Discussions* 2024, pp. 1–27. DOI: 10.5194/nhess-2024-26. URL: https://nhess.copernicus.org/preprints/nhess-2024-26/.

Haskins, Jason et al. (2020). "Exploring VR Training for First Responders". In: *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 57–62. DOI: 10.1109/VRW50115.2020.00018.

Haynes, Paul, Sigrid Hehl-Lange, and Eckart Lange (2018). "Mobile Augmented Reality for Flood Visualisation". In: *Environmental Modelling & Software* 109, pp. 380–389. ISSN: 1364-8152. DOI: https://doi.org/10.1016/j.envsoft.2018.05.012. URL: https://www.sciencedirect.com/science/article/pii/S1364815217302529.

Katsiokalis, Minas, Lemonia Ragia, and Katerina Mania (2020). "Outdoors Mobile Augmented Reality for Coastal Erosion Visualization Based on Geographical Data." In: *XR@ ISS*.

Kourogi, M. and T. Kurata (2003). "A wearable augmented reality system with personal positioning based on walking locomotion analysis". In: *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 2003. Proceedings.* Pp. 342–343. DOI: 10.1109/ISMAR.2003.1240751.

MacEachren, Alan M. et al. (2012). "Visual Semiotics & Uncertainty Visualization: An Empirical Study". In: *IEEE Transactions on Visualization and Computer Graphics* 18.12, pp. 2496–2505. DOI: 10.1109/TVCG.2012.279.

Mol, Jantsje M., W. J. Wouter Botzen, and Julia E. Blasch (2022). "After the virtual flood: Risk perceptions and flood preparednessafter virtual reality risk communication". In: *Judgment and Decision Making* 17.1, 189–214. DOI: 10.1017/S1930297500009074.

Nelson, Cassidy R. et al. (2022). "User-Centered Design and Evaluation of ARTTS: an Augmented Reality Triage Tool Suite for Mass Casualty Incidents". In: *2022 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 336–345. DOI: 10.1109/ISMAR55827.2022.00049.

Nescher, Thomas and Andreas Kunz (2012). "Analysis of Short Term Path Prediction of Human Locomotion for Augmented and Virtual Reality Applications". In: *2012 International Conference on Cyberworlds*, pp. 15–22. DOI: 10.1109/CW.2012.10.

Nunes, Isabel L et al. (2019). "An augmented reality application to support deployed emergency teams". In: *Proceedings of the 20th Congress of the International Ergonomics Association (IEA 2018) Volume V: Human Simulation and Virtual Environments, Work With Computing Systems (WWCS), Process Control 20*. Springer, pp. 195–204.

Oliveira, Natália Ferreira et al. (2015). "Uncertainty Visualization Framework for Improving Situational Awareness in Emergency Management Systems". In: *Human Interface and the Management of Information. Information and Knowledge Design*. Ed. by Sakae Yamamoto. Cham: Springer International Publishing, pp. 86–96. ISBN: 978-3-319-20612-7.

Potter, Kristin, Paul Rosen, and Chris R Johnson (2012). "From quantification to visualization: A taxonomy of uncertainty visualization approaches". In: *Uncertainty Quantification in Scientific Computing: 10th IFIP WG 2.5 Working Conference, WoCoUQ 2011, Boulder, CO, USA, August 1-4, 2011, Revised Selected Papers*. Springer, pp. 226–249.

Rae Nelson, Cassidy et al. (2022). "Exploring augmented reality triage tools to support mass casualty incidents". In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*. Vol. 66. SAGE Publications Sage CA: Los Angeles, CA, pp. 1664–1666.

Rydvanskiy, Ruslan and Nick Hedley (2021). "Mixed Reality Flood Visualizations: Reflections on Development and Usability of Current Systems". In: *ISPRS International Journal of Geo-Information* 10.2. ISSN: 2220-9964. DOI: 10.3390/ijgi10020082. URL: https://www.mdpi.com/2220-9964/10/2/82.

Sainidis, Dimitrios et al. (2021). "Single-handed gesture UAV control and video feed AR visualization for first responders". In: *Proceedings of the International Conference on Information Systems for Crisis Response and Management (ISCRAM), Blacksburg, VA, USA*, pp. 23–26.

Sarri, Froso et al. (2022). "Location-Aware Augmented-Reality for Predicting Sea Level Rise in Situ". In: *2022 International Conference on Interactive Media, Smart Systems and Emerging Technologies (IMET)*, pp. 1–8. DOI: 10.1109/IMET54801.2022.9929635.

Sebillo, Monica et al. (2015). "The use of augmented reality interfaces for on-site crisis preparedness". In: *Learning and Collaboration Technologies: Second International Conference, LCT 2015, Held as Part of HCI International 2015, Los Angeles, CA, USA, August 2–7, 2015, Proceedings 1*. Springer, pp. 136–147.

Sebillo, Monica et al. (2016). "Training emergency responders through augmented reality mobile interfaces". In: *Multimedia Tools and Applications* 75, pp. 9609–9622.

Sermet, Yusuf and Ibrahim Demir (Aug. 2018). "Flood Action VR: A Virtual Reality Framework for Disaster Awareness and Emergency Response Training". In: ISBN: 978-1-4503-6314-3. DOI: 10.1145/3306214.3338550.

Siu, Teresa and Valeria Herskovic (2013). "SidebARs: Improving awareness of off-screen elements in mobile augmented reality". In: *Proceedings of the 2013 Chilean Conference on Human-Computer Interaction*, pp. 36–41.

Smit, B.-P., R. Voûte, and E. Verbree (2021). "CREATING 3D INDOOR FIRST RESPONDER SITUATION AWARENESS IN REAL-TIME THROUGH A HEAD-MOUNTED AR DEVICE". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* V-4-2021, pp. 209–216. DOI: 10.5194/isprs-annals-V-4-2021-209-2021. URL: https://isprs-annals.copernicus.org/articles/V-4-2021/209/2021/.

Wang, Shanyu et al. (2020). "Visualization of Flood Simulation with Microsoft HoloLens". In: *Advances in Hydroinformatics: SimHydro 2019-Models for Extreme Situations and Crisis Management*. Springer, pp. 91–101.

Wani, Aameer (Nov. 2013). "Augmented Reality for Fire and Emergency Services". In.

Xu, Fang et al. (2024). "Augmented reality in team-based search and rescue: Exploring spatial perspectives for enhanced navigation and collaboration". In: *Safety Science* 176, p. 106556. ISSN: 0925-7535. DOI: https://doi.org/10.1016/j.ssci.2024.106556. URL: https://www.sciencedirect.com/science/article/pii/S0925753524001462.