Technical University of Crete

DIPLOMA THESIS

Design and Implementation of an Augmented Reality System for Navigation, Team Coordination, and Real-Time Data Visualization in Firefighting Operations

---

Author:
Charalampakis Konstantinos

Committee:
Mania Aikaterini
Deligiannakis Antonios
Giatrakos Nikolaos

A thesis submitted in fulfillment of the requirements
for the degree of Electrical and Computer Engineering
(2025)

Abstract

This thesis explores the potential of augmented reality (AR) technology to enhance the safety and effectiveness of firefighters in emergency situations. The research focuses on the design and implementation of an AR system developed for Microsoft HoloLens 2, aimed at supporting firefighting team leaders during real-time wildfire emergency operations. The system integrates GPS positions of team members, real-time fire locations, a fire spread simulation, and weather information into an interactive AR interface that overlays vital information directly into the user's field of view.

The proposed system is built using Unity and the Mixed Reality Toolkit (MRTK3), and it includes features such as a dynamic user interface, a 3D map powered by Mapbox, routing and navigation functionality, and real-time monitoring of fire activity and weather data, by visualizing the data in the AR scene. The AR system is designed to enhance operational awareness and enable safer, more efficient management of wildfire emergencies.

The system was evaluated in collaboration with an experienced firefighter to assess its usability and effectiveness in a realistic environment. Initial feedback highlighted that the system's features could be useful in real-world scenarios to improve situational awareness and decision-making, along with some suggestions for further improvements.

Overall, this thesis demonstrates how AR technology can be leveraged to support critical decision-making and coordination during emergencies to improve the safety and efficiency of firefighting operations. The research contributes to the growing body of knowledge on AR applications in emergency management and provides a foundation for future developments in this field.

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 CREX Data 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 firefighting.

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

Wildfires represent one of the most challenging natural disasters due to their rapid spread, unpredictability, and the vast areas they can affect. Efficient emergency response in such scenarios demands enhanced situational awareness, rapid decision-making, and seamless team coordination. This thesis presents the design and implementation of an augmented reality (AR) system developed for Microsoft HoloLens 2, aimed at supporting firefighting team leaders during real-time wildfire emergency operations.

1.1.1 Main System Features

The proposed system integrates GPS data, real-time fire locations, weather information, and spatial awareness tools into an interactive AR interface that overlays vital information directly into the user's field of view. Developed in Unity using the Mixed Reality Toolkit (MRTK3), the system is intended to enhance operational awareness and enable safer, more efficient management of wildfire emergencies.

Key components of the system include:

- A dynamic and customizable user interface, with hand-following menus and panels for map visualization, weather monitoring, and system setup.
- A 3D map powered by Mapbox, anchored to the user's real-world location, featuring zoom, repositioning, satellite/street view toggling, and target area selection.
- Visualization of team member locations in both the 2D map and the AR scene, with real-time updates based on GPS data transmitted via a Node.js server and WebSocket communication.

- Routing and navigation functionality that provides optimal paths to teammates or critical points of interest (POIs), displayed on both the map and through an AR-guided arrow system.
- Real-time monitoring of fire activity using satellite data from NASA FIRMS, including both current fire locations and a simulated fire spread visualization.
- Weather data integration from the OpenMeteo API, including temperature, wind speed/direction, and forecast uncertainty, along with AR-based wind direction indicators.
- User-driven calibration and GPS pairing with mobile devices to align the AR scene with real-world orientation, ensuring accurate spatial representation.

Figure 1.1: Testing the system

The system was tested in collaboration with an experienced firefighter to evaluate its usability and effectiveness in a realistic environment. Initial feedback emphasized the importance of satellite imagery, localized weather forecasting, and enhanced interaction methods for high-stress conditions.

This thesis explores the underlying technologies, the development process, and the integration challenges involved in building this AR system. It aims to demonstrate how immersive technology can be leveraged to support critical decision-making and coordination during natural disasters.

1.2 Structure of the Thesis

This thesis is organized into seven main chapters, plus a bibliography. Each chapter builds on the previous ones to guide the reader from the high-level motivation down to implementation details, evaluation, and future work:

- **Chapter 1: Introduction.** In this chapter, the motivational reasons for this research are introduced, the problem context is outlined, and the project's purpose and key contributions are summarized.
- **Chapter 2: Background and Research Overview.** In this chapter, the foundational protocols, platforms, and services supporting the system are surveyed, including GPS for positioning, WebSocket for real-time communication, Kafka for message brokering, Node.js for server-side development, and external APIs (Mapbox, NASA FIRMS, OpenMeteo). Augmented Reality fundamentals, HCI considerations, and fire-management use cases are also presented.
- **Chapter 3: Technological Background and Definitions.** In this chapter, the primary tools and frameworks employed in the implementation are introduced: the Unity game engine's AR capabilities, the Microsoft HoloLens 2 hardware platform and its strengths and limitations, the Mixed Reality Toolkit (MRTK3), and the use of Mapbox for 3D mapping.
- **Chapter 4: Server and Data Communication.** In this chapter, the backend server design and data-flow architecture are described, explaining how user devices connect, how GPS and sensor data are transmitted via WebSockets and Kafka, and the strategies for reliable synchronization and reconnection.
- **Chapter 5: Implementation.** In this chapter, the end-to-end system design and implementation are detailed, covering the high-level architecture, the AR user interface and interaction methods (gestures, gaze, voice), and the functional modules: location calibration, 3D map integration, routing and navigation, team tracking, points of interest, fire monitoring and spread simulation, and weather data visualization.
- **Chapter 6: Evaluation.** In this chapter, the evaluation process is reported, presenting local field trials and professional user feedback, discussing observed challenges, performance metrics, usability findings, and the iterative improvements they informed.
- **Chapter 7: Conclusion and Future Work.** In this chapter, the research contributions are summarized, limitations are reflected upon, and directions for future extensions and enhancements are outlined.

Chapter 2

Background/Research Overview

2.1 Introduction

This chapter lays the foundation for the development of our augmented reality system for real-time wildfire management. We begin by surveying traditional tools and emerging digital solutions used in firefighting operations, then introduce the key network protocols, software services, and hardware platforms that enable our system. Finally, we outline the structure of this chapter, which covers communication protocols (Section **??**), external API integrations (Section **??**), AR fundamentals and HCI considerations (Section **??**), and current applications of AR in fire management (Section **??**).

2.2 Augmented Reality

2.2.1 Introduction to AR

Augmented Reality (AR) is an interactive technology that enhances the physical world by overlaying digital elements such as images, videos, 3D models, audio cues, and real-time data onto a user's real-time view of their environment. These augmentations are typically perceived through devices such as smartphones, tablets, head-mounted displays (HMDs), or AR smart glasses. Unlike Virtual Reality (VR), which replaces the user's surroundings with a fully synthetic digital world, AR integrates virtual content seamlessly with the real world, maintaining the user's spatial and sensory awareness. This blending allows users to interact simultaneously with both physical and digital objects in a coherent space, often in real time.

AR relies on a combination of sensors, computer vision, simultaneous localization and mapping (SLAM), and environmental tracking to accurately position virtual objects within a user's physical surroundings. For example, an AR application

might detect horizontal surfaces (like tables or floors), track a user's hand gestures, and respond to voice commands, making the interaction highly intuitive and immersive.

Today, AR plays an increasingly important role across multiple domains. In education, AR transforms traditional textbooks into interactive learning tools. In healthcare, it assists in pre-surgical planning and real-time navigation. Retailers use AR to enable virtual try-ons or preview furniture placement within a customer's home. In industrial settings, AR supports complex assembly tasks with real-time visual instructions. The broad adaptability and spatial awareness of AR make it a transformative interface between humans and digital information.

Figure 2.1: Difference between AR and VR

2.2.2 Brief History of AR

Early Beginnings The conceptual foundation of AR dates back to 1968 when computer scientist Ivan Sutherland developed the first head-mounted display system, often referred to as the "Sword of Damocles." This device projected simple wireframe graphics onto the user's view, laying the groundwork for future AR systems.

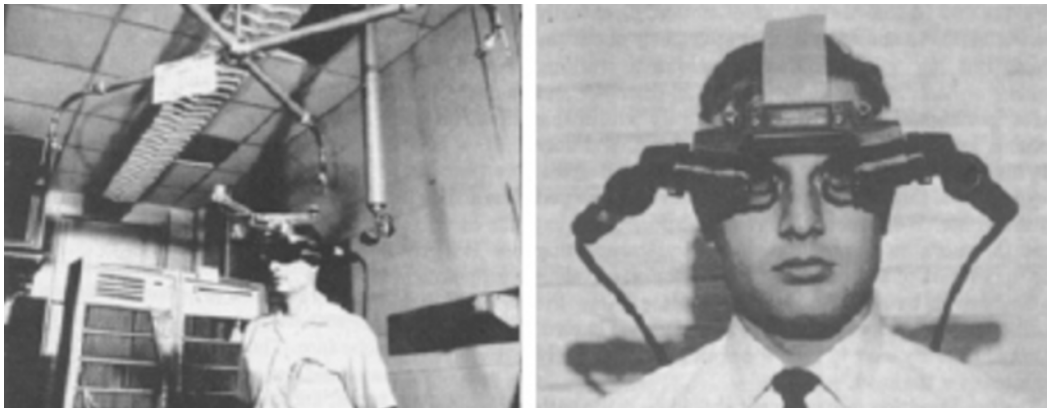

Figure 2.2: The Sword of Damocles, the first head-mounted display system

Development through the 1980s–1990s In the early 1990s, the term "Augmented Reality" was coined by Boeing researcher Tom Caudell to describe a digital display system designed to guide workers through aircraft assembly processes. Concurrently, Louis Rosenberg developed the "Virtual Fixtures" system for the U.S. Air Force, which allowed users to interact with virtual overlays in real-world tasks, marking a significant advancement in AR technology.

Commercialization Era (2000s–2010s) The 2000s witnessed AR's transition from research labs to commercial applications. Notably, the 2016 release of Pokémon GO demonstrated AR's potential in mobile gaming, engaging millions of users worldwide. In 2017, Apple and Google launched ARKit and ARCore, respectively, providing developers with robust platforms to create AR experiences on iOS and Android devices.

Modern AR: Current Platforms and Devices Today, AR technology is more accessible and sophisticated than ever. Leading platforms include Apple's ARKit, Google's ARCore, Microsoft's HoloLens, and Magic Leap's AR headsets. These platforms support a range of applications, from industrial training to interactive gaming. The integration of AR into smartphones and the development of lightweight, wearable AR devices have expanded its reach and usability, making it a valuable tool in various fields.

AR in Today's World AR has become an integral part of various industries. In retail, it enables virtual try-ons and product visualization, enhancing customer engagement. In healthcare, AR assists in surgical planning and medical training, improving precision and outcomes. Educational institutions utilize AR to create immersive learning experiences, making complex subjects more accessible. As AR technology continues to evolve, its applications are expected to expand further, transforming how we interact with the world around us and enhancing our daily lives.

2.2.3 Use Cases of Augmented Reality (AR)

Industrial and Manufacturing Applications AR systems are widely implemented on factory floors and in production environments to overlay assembly instructions, part numbers, operating parameters, and safety warnings directly onto physical machinery and components. Technicians equipped with AR headsets (e.g., Microsoft HoloLens, Magic Leap) can visualize step-by-step workflows projected in their line of sight, allowing them to carry out complex assembly, maintenance, or repair procedures without referring to manuals or screens. This hands-free access to dynamic, contextual information reduces cognitive load, minimizes human error, and accelerates task completion. Moreover, remote experts

can collaborate in real time by annotating the technician's view, supporting remote diagnostics and repairs. AR has also proven effective in warehouse management and quality assurance, where it guides workers to specific items or flags deviations from standard configurations.

Figure 2.3: Augmented Reality in industrial Applications

Education and Learning Environments In academic and museum settings, AR is used to convert static visual or textual material into immersive, interactive content. By using smartphones, tablets, or AR glasses, students can point at textbook pages, posters, or physical models to trigger 3D visualizations, simulations, or guided walkthroughs. Examples include animated depictions of biological systems, interactive molecular models, or reconstructions of historical landmarks and events. This multisensory approach to learning has been shown to increase student motivation, improve retention, and support individualized learning paths. In museums and cultural institutions, AR allows visitors to explore artifacts in their original context, such as viewing ancient ruins with reconstructed overlays of how they once appeared.

Healthcare and Medical Training AR is transforming both clinical practice and medical education by enabling richer, more precise visualizations of anatomical structures and real-time surgical contexts. Surgeons and trainees use AR to project three-dimensional medical data such as CT, MRI, or ultrasound scans directly onto a patient's body, enabling enhanced spatial understanding during diagnostics, planning, and procedures. AR-assisted surgery platforms (e.g., AccuVein for vein visualization or Proximie for remote surgery collaboration) provide

critical overlays that improve targeting and reduce invasiveness. In training scenarios, AR simulators allow students to practice procedures virtually on life-sized holographic patients, offering risk-free, repeatable environments for learning complex techniques, including emergency response.

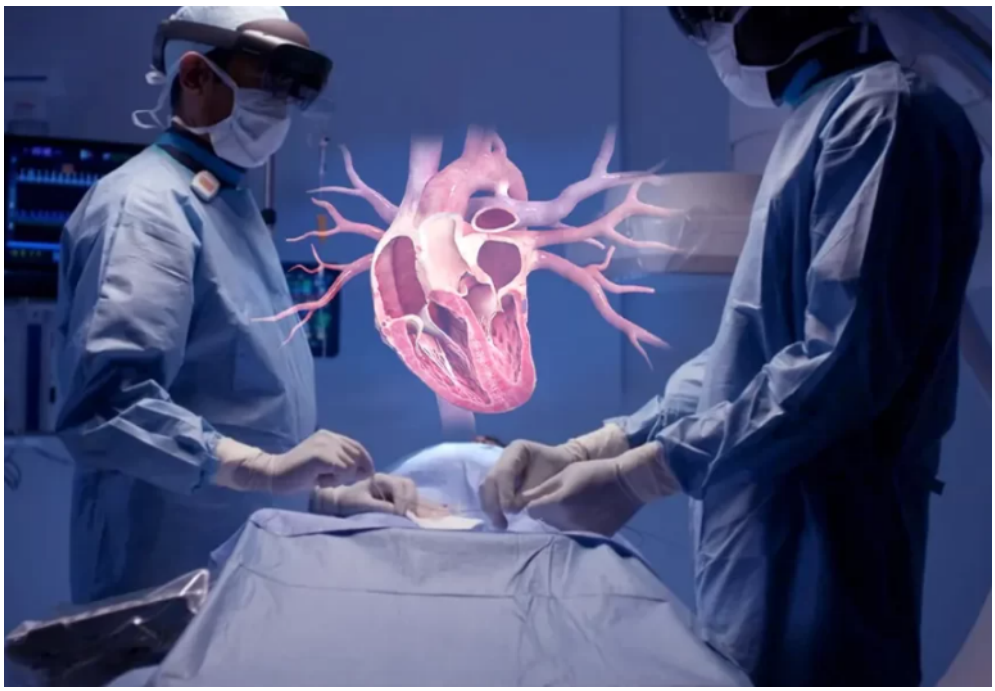

Figure 2.4: Augmented Reality in Healthcare

Disaster Management and Emergency Response AR is increasingly adopted in disaster preparedness, firefighting, and emergency medical response, where situational awareness and fast decision-making are critical. AR-enabled helmets and smart visors provide live overlays of building blueprints, evacuation routes, and hazardous material locations, assisting first responders navigating dangerous or unfamiliar environments. Integrated with IoT sensor data, AR systems can display real-time temperature, air quality, or structural integrity readings directly into the responder's field of view. This enhances coordination and reduces the risk of injury or loss of life. During training exercises, AR can simulate disaster scenarios (e.g., fires, floods, chemical spills) with high realism, preparing teams more effectively for real-world operations.

Entertainment and Gaming The entertainment industry has been one of the earliest adopters of AR, using it to create engaging, location-aware experiences that blend the virtual and physical worlds. AR games like *Pokémon GO*, *Ingress*, and *Harry Potter: Wizards Unite* have demonstrated the mass appeal of interacting with virtual elements in real-world locations. These games utilize GPS, cameras, and real-time rendering to allow users to "catch" characters, complete challenges, and explore surroundings with a gamified layer. Theme parks and live

events also use AR to enhance storytelling and immersion, such as overlaying digital creatures or special effects on stage performances or rides. Furthermore, social media platforms integrate AR for filters and effects, allowing users to creatively modify their environment or appearance in real time.

2.2.4 Human–Computer Interaction (HCI) in AR

Interaction Methods AR interfaces support multiple interaction paradigms, including:

- **Gesture Recognition**: Hand and body gestures detected by depth sensors or cameras enable users to manipulate virtual objects without physical controllers.
- **Voice Commands**: Natural language processing allows hands-free control of AR applications, useful in sterile or multitasking environments.
- **Eye Tracking**: Gaze-based selection and focus control facilitate intuitive targeting and menu navigation.
- **Haptic Feedback**: Vibrotactile gloves or wearable devices provide tactile sensations when interacting with virtual elements, improving immersion and task precision.

Ergonomics and User Experience (UX) in AR Effective AR UX design must consider display weight, field of view, and battery life to minimize user fatigue. Transparent optics should balance opacity for virtual content against visibility of the real world to prevent motion sickness. Interfaces need clear affordances visual cues for interaction zones and state changes to maintain user confidence in mixed-reality tasks.

Adaptive Interfaces and Context–Awareness Context-aware AR systems leverage sensors (GPS, IMU, camera) and machine learning algorithms to adapt content based on user location, activity, and environmental conditions. For example, industrial AR may highlight only relevant components during a maintenance task, while retail AR might display promotional information when a user points at a product. Adaptive interfaces reduce information overload and tailor experiences to individual needs and contexts.

2.3 Augmented Reality in Fire Management

2.3.1 Introduction

Augmented reality (AR) technology holds the potential to substantially improve the safety and effectiveness of firefighters during emergency situations[16, 19,

13, 18]. By overlaying critical information onto the user's view of the physical world, AR can enhance situational awareness and decision-making capabilities within hazardous environments. This section examines the current state of the art in augmented reality applications specifically for firefighting, highlighting recent advancements and identifying areas requiring further research.

2.3.2 Background

Firefighting is an inherently dangerous profession, presenting numerous challenges such as low visibility due to smoke, extreme temperatures, and potential disorientation caused by debris and unfamiliar layouts. Maintaining situational awareness is therefore critical for firefighter safety and operational success. Traditional methods for navigation and hazard detection rely heavily on the experience and intuition of firefighters, capabilities which can be severely compromised under the extreme stress and conditions of a fire scene. Recent advancements in augmented reality offer novel ways to assist firefighters. AR systems can provide hands-free access to vital information—such as building schematics, the real-time location of team members, and identified hazards—directly within their field of view, potentially mitigating the limitations of traditional methods in high-stress environments.

2.3.3 Key AR Technologies and Systems

The implementation of AR in firefighting leverages several key technologies:

- **AR Displays:** Head-mounted displays (HMDs) like the Microsoft HoloLens allow for the projection of processed data and visual cues directly into the firefighter's line of sight. This enables the visualization of critical elements such as escape routes, structural features (doors, windows), or known hazards that might otherwise be obscured by smoke or darkness [1, 7, 16].
- **Thermal Imaging Integration:** AR systems frequently integrate thermal imaging cameras. By mapping infrared data onto the visible spectrum displayed via the HMD, these systems help firefighters perceive heat signatures through smoke, identify hotspots, locate victims, and potentially detect hidden structural weaknesses [10, 17].
- **Navigation and Communication Tools:** Enhanced navigation is a core benefit. AR can display building layouts, pre-planned routes, or dynamically updated evacuation paths. Furthermore, displaying the locations of fellow team members improves coordination and safety. These systems often incorporate communication links, facilitating information exchange between field teams and incident command centers [7].
- **Image Recognition and Object Detection:** Advanced algorithms can identify and classify objects in the environment, such as relevant objects or

victims, and overlay this information onto the AR display[17]. This capability can be particularly useful in low-visibility conditions, where traditional visual cues may be obscured.

An example of an integrated platform is the ENGAGE Incident Management System, which utilizes AR for cross-device collaboration during wildfire, urban, and rescue missions. It enhances command's situational awareness by presenting firefighter locations on a 3D map, using real-time tracking data [7]. Research has also explored adapting AR magnification techniques, originally developed for low-vision users, to potentially improve firefighter visibility in smoke-filled environments [2].

2.3.4 Related Work

Research in AR for firefighting spans hardware design, software integration, and human-computer interaction. Several studies provide context on the broader use of these technologies in emergency management. Foundational work has explored the concept of 'Intelligent Firefighting' which integrates various technologies to enhance response capabilities [9].

Significant research effort has focused on developing specialized hardware. This includes the design of compact near-eye displays suitable for integration into Self-Contained Breathing Apparatus (SCBA) masks [5] and concepts for intelligent firefighter helmets incorporating multiple sensors and AR displays [6]. Smart helmet prototypes combining sensors, Artificial Intelligence (AI), AR, and personal protective equipment aim to provide comprehensive situational awareness enhancements for first responders [12]. Wearable interfaces and advanced sensors are also being explored to specifically enhance safety during demanding scenarios like forest fires [14]. Early work highlighted key technological and human factors influencing the adoption of wearable computers in emergency services [15].

Another key area involves integrating sensor data and intelligent algorithms within AR systems. Studies have demonstrated the feasibility of using embedded deep learning for AR applications in firefighting [1], potentially enabling on-device hazard recognition or navigation assistance. The integration of real-time data from sensor networks is emphasized for robust scene perception and navigation through smoke [3, 10]. Evaluations comparing depth and thermal cameras have informed how best to augment firefighter visual perception within AR interfaces [11]. Furthermore, cross-device AR systems integrating thermal feeds and live tracking have been developed to provide comprehensive operational insights during fire and rescue operations [7, 10].

Understanding the human element is crucial for successful AR implementation. Iterative user studies focusing on augmented cognition have investigated how AR can support cognitive functions and improve decision-making during fire emergencies [2, 8]. User-centered design methods, sometimes employing virtual reality (VR) for rapid prototyping, have been used to develop and refine AR tools

specifically for firefighters [8]. Research also explores combining AR and VR for immersive training experiences, such as fire drills [4, 21, ?].

Overall, the related work highlights a multi-faceted approach to leveraging AR, focusing on hardware development, sophisticated data integration, and user-centered design to meet the demanding requirements of firefighting operations.

2.4 Protocols and Services

2.4.1 Global Positioning System (GPS)

The Global Positioning System (GPS) is a satellite-based service originally developed by the U.S. Department of Defense and now freely available for civilian use. It provides accurate latitude, longitude, altitude, velocity and precise time to any receiver on or near Earth, given unobstructed line of sight to at least four orbiting satellites.

GPS operates via a constellation of at least 24 medium-Earth-orbit satellites, each broadcasting its precise orbital data (ephemeris) and time of transmission. A GPS receiver captures signals from multiple satellites, compares the transmission time to its own clock, and computes the distance to each satellite. Using trilateration, the receiver derives its three-dimensional position and corrects any clock error by incorporating a fourth satellite's measurement. The result is a continuous, real-time position fix, updated multiple times per second.

The concept of satellite navigation dates back to the 1960s with the U.S. Navy's TRANSIT system for submarines. In 1973, the U.S. Department of Defense initiated the Navstar GPS program to achieve global coverage and precise timing. The first GPS satellite launched in 1978, and by 1995 the full 24-satellite constellation was operational. Civilian access expanded throughout the 1980s, and in 2000 the deliberate degradation of civilian signals (Selective Availability) was discontinued, improving public accuracy from roughly 100 m to under 10 m. Today's GPS benefits from modernized signals (e.g. L2C, L5) and regional augmentation systems (WAAS, EGNOS), offering sub-meter precision and enhanced reliability.

General GPS Use Cases

- **Navigation & Transportation:** In-car guidance, maritime and aviation routing, fleet tracking.
- **Mapping & Surveying:** GIS, land surveying, urban planning.
- **Timing & Synchronization:** Financial networks, telecommunications, power-grid coordination.
- **Agriculture & Environment:** Precision farming, autonomous machinery, wildlife monitoring.

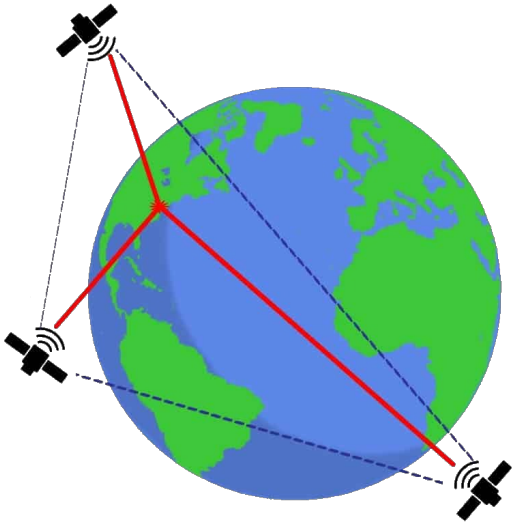

Figure 2.5: Satellite-based trilateration for GPS positioning

- **Emergency Response:** Dispatch, location tracking, search and rescue coordination.

In this work, GPS underpins both mapping and team coordination within our augmented reality wildfire management system. Each firefighter's smartphone app samples GPS coordinates at a configurable rate and transmits them, along with an identifier, to a central server. The Mapbox API then renders a responsive map in the AR interface, auto-centering and updating with the captain's position. Incoming GPS data are converted into Unity scene coordinates via geospatial transformation: by aligning the real-world origin with the virtual origin, we accurately place 3D markers for team members and critical points of interest. Continuous GPS updates drive dynamic routing overlays, displayed as paths on the map and as directional arrows in the AR world, enabling the team captain to monitor movements, plan routes, and coordinate resources safely and efficiently.

2.4.2 WebSocket Protocol

WebSocket is a protocol that enables full-duplex communication channels over a single TCP connection. It is designed to be implemented in web browsers and servers, providing a standardized way for real-time data exchange between clients and servers. WebSocket is particularly useful for applications that require low latency and high-frequency updates, such as online gaming, chat applications, live data feeds and collaborative tools.

WebSocket operates over the same ports as HTTP (port 80 for unsecured and port 443 for secured connections). The protocol begins with an HTTP handshake, where the client sends an HTTP request to the server to establish a WebSocket connection. If the server supports WebSocket, it responds with an HTTP 101

status code, indicating that the protocol is switching from HTTP to WebSocket. Once established, the connection remains open, allowing both parties to send and receive messages at any time without the overhead of repeated HTTP requests.

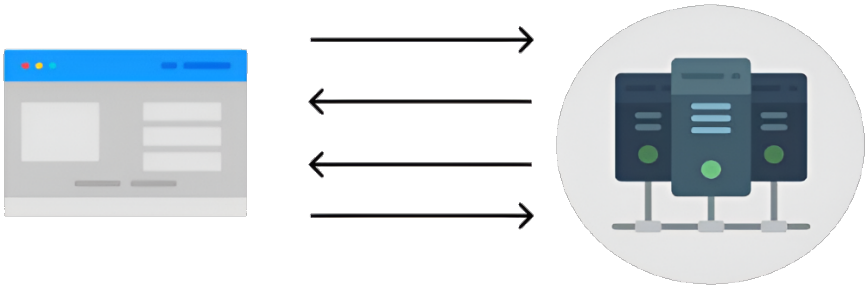

Figure 2.6: WebSocket Protocol

The WebSocket protocol was standardized by the IETF as RFC 6455 in 2011. It was developed to address the limitations of traditional HTTP communication, particularly for applications requiring real-time updates and bidirectional communication. Prior to WebSocket, developers relied on techniques like long polling or server-sent events (SSE) to achieve similar functionality, but these methods were often inefficient and introduced significant latency. Since its introduction, WebSocket has gained widespread adoption in web applications, enabling developers to create more interactive and responsive user experiences.

Typical Uses of WebSocket

WebSocket is commonly used in various applications that require real-time communication, including:

- **Online Gaming:** Multiplayer games use WebSocket to synchronize game state and player actions in real time, providing a seamless gaming experience.
- **Chat Applications:** WebSocket enables instant messaging and notifications, allowing users to send and receive messages without refreshing the page.
- **Live Data Feeds:** Financial applications, sports updates, and news aggregators use WebSocket to deliver real-time data streams to users.
- **Collaborative Tools:** WebSocket is used in applications like Google Docs or Trello, where multiple users can edit documents or boards simultaneously, with real-time updates reflecting changes made by others.

In this thesis, WebSocket is used to facilitate real-time communication between the server and client applications. The server receives GPS data from each firefighter's mobile device and broadcasts it to all connected clients, ensuring that

the team captain has up-to-date information on the location of each team member. This enables efficient coordination and decision-making during emergency operations.

2.4.3 JavaScript Object Notation (JSON)

JSON (JavaScript Object Notation) is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. It is based on a subset of the JavaScript programming language and is widely used for transmitting data between a server and a web application as an alternative to XML.

JSON is language-independent, meaning it can be used with various programming languages, including JavaScript, Python, Java, C#, and many others. It is often used in web APIs to exchange data between a client and a server.

In JSON data is represented as key-value pairs, similar to dictionaries or hash tables in other programming languages. The format supports various data types, including strings, numbers, booleans, arrays, and objects. This flexibility makes JSON a popular choice for representing complex data structures in a simple and readable format.

In this system, JSON is used to format the data exchanged between the server and client applications. For example the gps data, the weather data, the fire locations and the fire spread simulation data are all transmitted in JSON format. This allows for easy parsing and manipulation of the data on both the server and client sides.

2.4.4 Kafka

Kafka is an open-source distributed event streaming platform designed for high-throughput, fault-tolerant, and scalable data processing. Originally developed by LinkedIn and later donated to the Apache Software Foundation, Kafka has become a widely adopted solution for building real-time data pipelines and streaming applications. It is particularly well-suited for scenarios where large volumes of data need to be ingested, processed, and analyzed in real time, such as log aggregation, event sourcing, and stream processing.

Kafka's architecture is based on a publish-subscribe model, where producers publish messages to topics, and consumers subscribe to those topics to receive the messages. Each topic is divided into partitions, allowing for parallel processing and scalability. Kafka's distributed nature ensures that data is replicated across multiple brokers, providing fault tolerance and high availability.

Kafka's core components include:

- **Producers:** Applications or services that publish messages to Kafka topics.

- **Consumers:** Applications or services that subscribe to Kafka topics and process the messages.
- **Brokers:** Kafka servers that store and manage the data. A Kafka cluster consists of multiple brokers for scalability and fault tolerance.
- **Topics:** Categories or feeds to which messages are published. Each topic can have multiple partitions for parallel processing.
- **Zookeeper:** A distributed coordination service used by Kafka to manage cluster metadata, leader election, and configuration management.

In this system, Kafka enables efficient and reliable communication between the system and the data sources, such as fire spread simulation data, enviromental data, and alert notifications related to operation. Kafka's ability to handle high-throughput data streams and provide fault tolerance makes it an ideal choice for building a robust and scalable architecture for the augmented reality system. By decoupling the data producers and consumers, Kafka allows for flexible integration of various components, enabling real-time processing and analysis of the data.

2.5 External Services & APIs

2.5.1 Mapbox

Mapbox is a cloud-based mapping and location data platform that offers developers a comprehensive suite of tools to create, customize, and deploy interactive maps for web, mobile, and embedded applications. Founded in 2010 in San Francisco as part of the nonprofit mapping consultancy Development Seed. Mapbox emerged to address the limitations of legacy mapping services by providing highly customizable vector maps, open-source tooling, and scalable tile infrastructure. Early projects focused on improving OpenStreetMap (OSM) data quality, and the company's initial offerings included the TileMill map design studio and the iD editor for direct OSM contributions.

Figure 2.7: Mapbox Logo

Core Architecture Mapbox's platform is built around three foundational components:

- **Vector Tile Rendering.** Mapbox GL JS and native SDKs (iOS, Android, Qt) render maps client-side using WebGL. This approach allows smooth transitions, dynamic styling, and precise control over each map element at arbitrary zoom levels without requiring new raster tile downloads.
- **Data Ingestion & Processing.** Raw geospatial data from OpenStreetMap, NASA, and proprietary imagery (e.g., DigitalGlobe) are processed via Mapnik, GDAL, and a Node.js-based pipeline. The resulting vector tiles are stored in tile servers and served through a global CDN.
- **Telemetry-Driven Updates.** Anonymized user telemetry from partners such as Strava and Runkeeper is analyzed to detect missing roads or POIs. Automated scripts generate OSM edits or notify local contributors, ensuring that the map reflects real-world changes with low latency.

Main Use Cases

1. *Custom-Themed Maps.* Companies like Lonely Planet and The Weather Channel use Mapbox to deliver branded map experiences that align with their visual identity and content.
2. *Real-Time Navigation.* Ride-sharing and delivery platforms embed the Directions API to provide drivers with optimized routes and dynamic ETAs.
3. *Location Search and Discovery.* Retailers and travel apps integrate the Geocoding API and Search SDK to help users find nearby stores, restaurants, or attractions.
4. *Data Visualization.* Urban planners and analytics dashboards leverage vector tiles to render heat maps, flow maps, and animated overlays of mobility data.
5. *Indoor Mapping.* Airports, shopping malls, and event venues map interiors at high resolution, enabling asset tracking, wayfinding, and location-based notifications.

Example Projects

- **Strava Global Heatmap.** Aggregates billions of anonymized GPS traces from runners and cyclists into a multilayered heatmap, allowing users to discover popular routes and analyze activity patterns worldwide.
- **Foursquare City Guide.** Powers the interactive map interface in the Foursquare app, displaying venues, tips, and user check-ins with custom styling and fast POI search via the Geocoding and Search APIs.

- **The Weather Channel.** Integrates Mapbox vector tiles to render seamless, animated weather overlays, such as radar, precipitation, and temperature gradients on both web and mobile platforms.

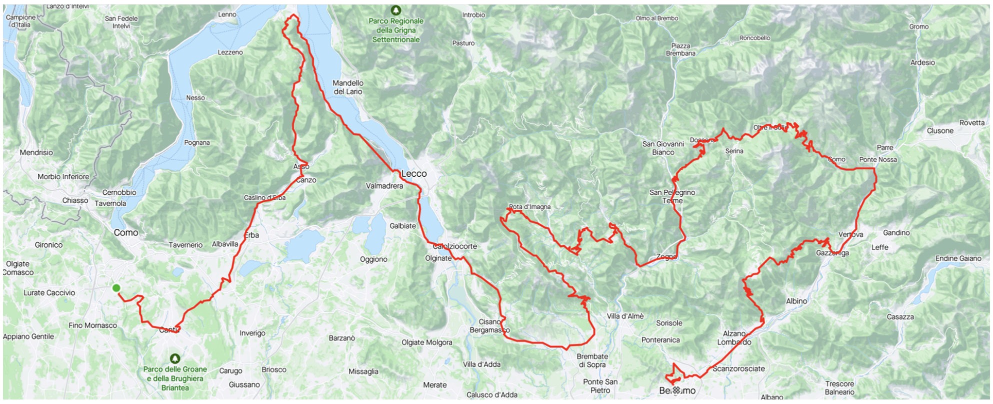

Figure 2.8: Strava-generated map using Mapbox

Mapbox has transformed digital cartography by coupling open-source principles with enterprise-grade infrastructure. With a history rooted in community-driven data, a robust vector rendering engine, and a versatile suite of APIs, Mapbox empowers developers to embed rich, responsive maps into virtually any application. From powering consumer hits like Snap Map to enabling mission-critical logistics and urban planning solutions, Mapbox continues to define the frontier of location-based services.

In this work, we use Mapbox to provide a responsive map interface for the augmented reality system, based on the user's location, were all the main information is displayed. The mapbox API is also used to handle the navigation and routing functionalities, by providing the best route to the user based on the current location and the destination.

2.5.2 NASA FIRMS

Introduction The NASA Fire Information for Resource Management System (FIRMS) is a publicly accessible web service that provides near–real-time data on active fire detections around the globe. Developed by NASA's Goddard Space Flight Center, FIRMS ingests satellite observations to identify thermal anomalies commonly wildfires, agricultural burning, or industrial heat sources and delivers these as geolocated "fire pixels" via a simple RESTful interface.

Brief History FIRMS was launched in 2012 to democratize access to fire monitoring data derived from NASA's Moderate Resolution Imaging Spectroradiometer (MODIS) instruments aboard the Terra and Aqua satellites. In 2013, FIRMS expanded to include data from the Visible Infrared Imaging Radiometer Suite

Figure 2.9: NASA FIRMS

(VIIRS) on the Suomi NPP and NOAA-20 platforms, offering higher spatial resolution and improved detection of smaller or lower-temperature fires. Over time, the API has evolved to support multiple output formats, customizable queries, and integration with geospatial platforms.

API Functionality The FIRMS API enables users to query, filter, and retrieve active fire data through a set of endpoints that return results in formats such as GeoJSON, CSV, KML, or plain text. Core features include:

- **Temporal Filtering.** Specify start and end dates (UTC) to retrieve fire detections within a given time window, from the last few hours up to historical archives.
- **Spatial Constraints.** Define geographic bounding boxes or polygonal regions (latitude–longitude pairs) to limit queries to areas of interest, such as countries, states, or custom shapes.
- **Sensor Selection.** Choose between MODIS (1 km resolution, four updates per day) and VIIRS (375 m resolution, two updates per day) datasets, balancing coverage frequency and spatial detail.
- **Confidence Filtering.** Filter detections by confidence level (low, nominal, high) to exclude uncertain or spurious anomalies.
- **Magnitude Data.** Each fire record includes brightness temperature, scan and track angles, and derived Fire Radiative Power (FRP), enabling users to assess fire intensity.
- **Multiple Output Formats.** Receive results as GeoJSON for web mapping, CSV for tabular analysis, KML for GIS applications, or plain text for custom parsing.

Data are typically available within three hours of satellite overpass, allowing near–real-time monitoring of fire outbreaks, smoke exposure forecasting, and resource management. By combining high-frequency satellite detections with flexible query parameters, the FIRMS API empowers researchers, emergency responders, and decision-makers to integrate active fire location data into applications ranging from wildfire tracking dashboards to mobile alerting systems.

In this system, we use the FIRMS Data to display the active fire locations on the map, providing the firefighters with a near real-time overview of the fire situation

in the area. The data retrieved from the NASA FIRMS are being displayed on the map and the AR scene, helping the firefighters to understand the current fire situation and make informed decisions about resource allocation and evacuation routes during emergency operations.

2.5.3 Spark Firespread Simulation Tool

The Spark is a fire simulation tool that allows users to model and analyze the spread of wildfires in real-time. It is designed to be flexible and extensible, enabling users to create custom fire propagation models based on their specific needs. The tool uses a computational fire propagation solver that can simulate the spread of fire over a wide range of conditions, including different fuel types, topography, and meteorological factors.

The Spark tool is built on the Geostack platform, which provides a foundation for geospatial modeling and simulation. It uses high-performance OpenCL technology to run simulations quickly and efficiently, making it suitable for both desktop and cluster computing environments. The underlying algorithm is based on the raster-based level set method, which allows for precise control over fire propagation.

The Spark tool supports various input and output data formats, including text files, CSV files, raster images, and ESRI formats. Users can define their own input layers, such as fuel parameters and meteorological conditions, and link them to the speed function that controls fire propagation. The tool also includes built-in modules for data analysis and visualization, allowing users to inspect point values, plot data, and perform statistical calculations over the simulation domain.

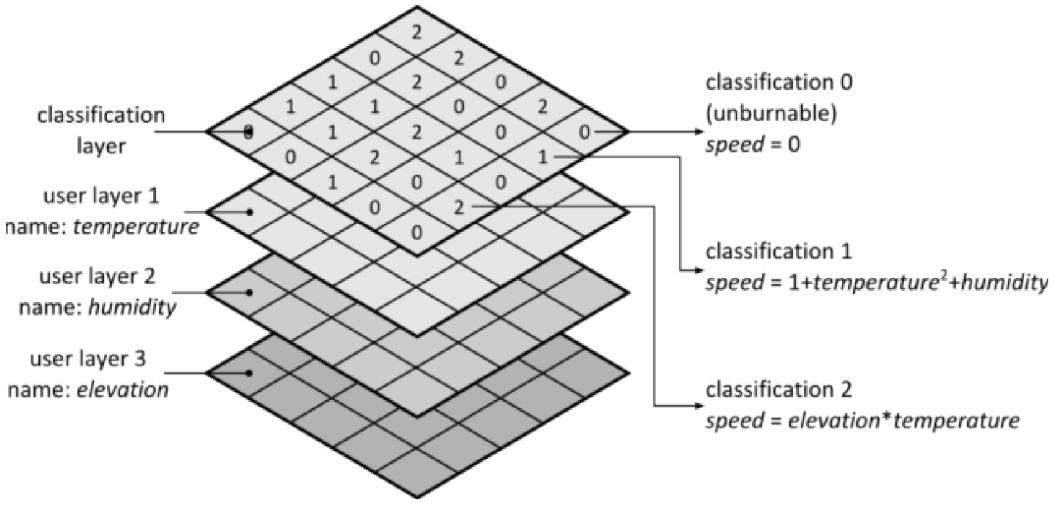

Figure 2.10: Spark Input Layers Example

In our system, we use the output from the Spark tool to visualize the fire spread simulation in real-time. The simulation results are integrated into the augmented

reality interface, providing firefighters with critical information about the fire's behavior and potential spread. This helps them make informed decisions about resource allocation and evacuation routes during emergency operations.

2.5.4 OpenMeteo API

OpenMeteo is a free weather API that provides weather data for any location worldwide. It offers a wide range of weather parameters, including temperature, precipitation, wind speed, humidity, and more. The API is designed to be easy to use and provides data in a simple JSON format.

OpenMeteo was launched in 2020 as an open-source weather API, aiming to provide accurate and reliable weather data without the need for complex authentication or payment. The API is built on top of various weather models and data sources, including global and regional weather forecasts, historical data, and satellite observations.

The API allows users to retrieve weather data for specific locations by providing latitude and longitude coordinates. Users can specify the desired time range and the weather parameters they want to receive. The API supports both current weather data and forecasts for up to 14 days in advance.

In our system, we use the OpenMeteo API to retrieve real-time weather data for the area where the firefighters are operating. This information is crucial for understanding the current weather conditions and how they may affect the fire's behavior. The weather data is integrated into the augmented reality interface, providing firefighters with current and forecasted weather conditions, such as temperature, humidity, wind speed, and direction. This helps them make informed decisions about resource allocation and evacuation routes during emergency operations.

2.5.5 Conclusion of Protocols and Services

In this chapter, we have discussed the key protocols and services that form the backbone of our augmented reality wildfire management system. These include GPS for location tracking, WebSocket for real-time communication, JSON for data interchange, and various external APIs such as Mapbox for mapping, NASA FIRMS for fire detection, Spark for fire spread simulation, and OpenMeteo for weather data.

These technologies were put together to create a comprehensive system that enhances situational awareness and decision-making for firefighters during emergency operations. By integrating real-time data from multiple sources, we aim to improve the safety and effectiveness of firefighting efforts in challenging environments.

Chapter 3

Technological Background and Definitions

3.1 Unity Engine

3.1.1 Introduction to Unity

Unity, created by Unity Technologies and first released in 2005, is a comprehensive, cross-platform game and real-time development engine. Originally conceived as an affordable tool for independent developers to build 3D games on the Mac OS X platform, Unity quickly expanded its reach to Windows in 2006 and has since grown to support over 25 target platforms, including desktop, mobile, web, console, XR, and embedded systems. Its rise to prominence was fueled by a user-friendly, easy to use editor that abstracts away low-level graphics programming while still exposing a powerful C# scripting API for customization and complex logic.

Figure 3.1: Unity Engine Logo

Historically, Unity democratized game creation by offering a free Personal Edition in 2009, followed by tiered Professional and Enterprise licenses to serve studios of all sizes. Major milestones include the introduction of the Asset Store in 2010, an online marketplace for assets, tools, and extensions, and the integration of

high-fidelity graphics pipelines (URP and HDRP) in 2018, which enabled scalable rendering from mobile GPUs to cutting-edge desktop hardware. Over two decades, Unity has cultivated a vast ecosystem of educators, artists, and engineers, supported by extensive documentation, tutorials, an active community forum, and annual developer conferences.

Today, Unity's combination of rapid prototyping capabilities, real-time rendering, and broad platform compatibility makes it a favored engine not only for video games but also for simulations, architectural visualization, interactive installations, and Augmented Reality (AR) applications. Its continual evolution embracing features like DOTS (Data-Oriented Tech Stack), GPU-accelerated compute, and integrated AI toolkits, ensures Unity remains at the forefront of real-time 3D development.

3.1.2 Core Features of Unity

- **Visual Editor & Scene Management.** A very comprehensible and easy to use interface for arranging scenes, game objects, prefabs, lighting, and UI elements with real-time previews.
- **Component-Based Architecture.** Reusable components attached to GameObjects allow modular behaviors and custom scripts written in C# extend its functionality.
- **Rendering Pipeline.** Built-in support for both the High Definition Render Pipeline (HDRP) and Universal Render Pipeline (URP) enables scalable graphics from mobile to high-end hardware.
- **Physics & Animation.** Integrated PhysX physics engine, cloth, soft bodies, and Mecanim animation system for character rigs, blend trees, and timeline sequencing.
- **Asset Management.** The Unity Asset Store offers thousands of plug-and-play assets—models, shaders, tools, and templates—accelerating prototyping and production.
- **Cross-Platform Build System.** One-click builds to over 25 platforms, with platform-specific optimizations and build scripting via the Editor and CLI.
- **Networking & Services.** Built-in support for multiplayer networking, analytics, cloud build, and performance reporting to streamline live operations.

3.1.3 Common Uses of Unity

Unity's flexibility makes it a go-to engine for a wide range of applications beyond traditional gaming:

- *Video Games:* Indie and high-quality video games from major studios, in 2D, 3D, and VR formats for PCs, consoles, and mobile.
- *Architectural Visualization:* Real-time walkthroughs and interactive floor plans for real estate and construction.
- *Training Simulations:* Military, aviation, medical, and industrial training through immersive scenario-based modules.
- *Film & Animation Previsualization:* Scene layout, camera blocking, and virtual cinematography prior to live-action shoots.
- *Interactive Installations:* Museum exhibits, live events, and art installations utilizing real-time graphics and user interaction.

Figure 3.2: Filmaking using Unity. Movie: Adam

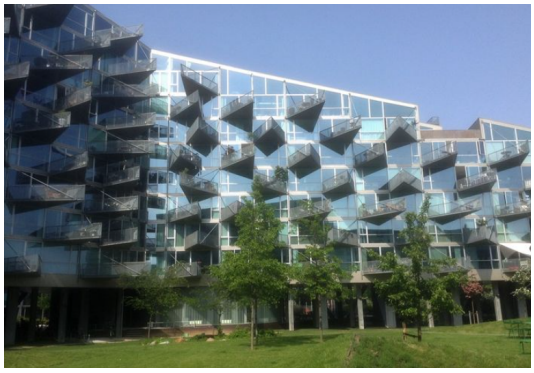

Figure 3.3: Architecture Visualization using Unity

3.1.4 Unity for Augmented Reality

Unity has become one of the most widely adopted platforms for developing AR applications today, powering everything from consumer mobile apps to enterprise mixed-reality solutions. Its AR Foundation framework unifies platform-specific SDKs (ARKit on iOS and ARCore on Android) under a single, high-level API, enabling developers to rapidly prototype and deploy across devices. This abstraction not only simplifies cross-platform development by providing consistent components and behaviors, but also leverages Unity's powerful editor tooling such as real-time scene previews, play-mode simulation of AR environments, and an extensive library of prefabs and shaders, to accelerate debugging and iteration. Integrated performance profiling, asset management, and extensibility via C# scripting further streamline the creation of sophisticated AR experiences.

Figure 3.4: Example: Placing AR Models into a Virtual Scene

When used in conjunction with ARKit, Unity provides users with a rich set of device-specific features, including:

- **Face Tracking & Blend Shapes.** Precise detection of facial landmarks to drive realistic avatar expressions or apply dynamic filters in real time.
- **Body Tracking & Motion Capture.** Full-body skeletal tracking for interactive fitness, gaming, or avatar control applications.
- **Collaborative Sessions.** Multi-user AR experiences that synchronize spatial anchors and content across devices for shared interactions.
- **Environment Probes & Occlusion.** Realistic integration of virtual objects by capturing ambient lighting and providing depth occlusion, so digital content appears properly behind or in front of real-world surfaces.
- **Scene Reconstruction & Mesh Generation.** Runtime generation of detailed spatial meshes from the physical environment, allowing content to interact with real geometry.
- **Location Anchors & Geo-AR.** Persistent, geographically anchored content that remains in place across sessions, enabling large-scale outdoor AR experiences.

For this thesis, Unity was selected as the core engine for building the HoloLens 2 application, owing to its extensive mixed-reality toolset and seamless cross-platform capabilities. By leveraging Unity's AR Foundation and XR Interaction Toolkit alongside the Mixed Reality Toolkit 3 (MRTK3), the application offers robust spatial mapping, gesture and voice interaction, and a consistent development workflow. Targeting the Universal Windows Platform (UWP) build ensures full compatibility with HoloLens 2 hardware, leverages the device's Holographic Processing Unit (HPU) for optimal performance, and streamlines deployment through Microsoft's Store ecosystem.

3.2 Microsoft HoloLens 2

Overview of Microsoft HoloLens

Microsoft HoloLens is a self-contained, untethered mixed-reality headset first announced in January 2015 and commercially released in March 2016. Building on a lineage of research into head-mounted displays and augmented reality dating back to the 1990s, HoloLens introduced a novel combination of see-through holographic lenses, spatial mapping, and voice/gesture interfaces. The original device demonstrated the feasibility of holographic computing, enabling users to place and interact with 3D digital content in their physical environment without external sensors or a tethered PC. In November 2019, Microsoft launched HoloLens 2, refining the hardware and software to address user feedback and expand enterprise adoption.

Figure 3.5: Example: Hololens2 device

HoloLens 2 Capabilities

- **Expanded Field of View.** Approximately double the visible holographic area compared to the first HoloLens, providing a more immersive experience.
- **Advanced Optics.** Waveguide-based lenses with 2k × 2k resolution per eye and 3D positional tracking for sharper, more realistic holograms.
- **Hand and Eye Tracking.** Fully articulated hand tracking for direct manipulation of holograms and integrated eye-tracking sensors for intuitive control and foveated rendering.
- **Spatial Mapping & Anchors.** Real-time scanning of the environment to generate detailed 3D meshes, enabling persistent placement of virtual objects across sessions.

- **Audio & Voice Recognition.** Spatial sound with built-in speakers and microphones, coupled with natural language processing via Microsoft's Speech SDK for voice commands.
- **Onboard Processing & Connectivity.** Qualcomm Snapdragon 850 compute platform with holographic processing unit (HPU), 4 GB RAM, Wi-Fi 5 and Bluetooth 5.0 for standalone operation.
- **Comfort and Ergonomics.** Adjustable headband, flip-up visor design, and balanced weight distribution for extended wear in enterprise settings.

Limitations of HoloLens 2

- **Field of View Constraints.** Although improved, the holographic viewport remains limited compared to natural human vision, potentially reducing immersion.
- **Battery Life.** Approximately 2–3 hours of active use, requiring downtime or external battery packs for prolonged sessions.
- **Outdoor Performance.** Reduced visibility of holograms in bright sunlight and environmental conditions, as well as limited spatial mapping accuracy at greater distances outdoors.

For this thesis project, the Microsoft HoloLens 2 was chosen as the target device due to its advanced mixed reality capabilities, standalone functionality, and rich user interface options. Its precise hand and eye tracking enable intuitive gaze-based UI navigation and gesture controls. The high-resolution holographic display and ergonomic design facilitate clear presentation of on-screen dashboards, interactive menus, and data overlays, making HoloLens 2 an ideal platform for crafting user-centric, immersive applications.

3.3 Mixed Reality Toolkit (MRTK3)

3.3.1 Introduction to MRTK3

The Mixed Reality Toolkit version 3 (MRTK3) is an open-source SDK and component library, first released in late 2021 as the successor to MRTK2. Co-maintained by Microsoft and the wider mixed reality community, MRTK3 provides a modular, extensible framework for building cross-platform MR applications in Unity. It supports a variety of devices including HoloLens 2, Windows Mixed Reality headsets, and any OpenXR-compatible hardware by unifying input handling, UI construction, and simulation tools under a consistent API. Designed to streamline development, MRTK3 offers optimized performance on HoloLens 2, leveraging the device's Holographic Processing Unit (HPU) for low-latency interactions and efficient resource management.

Figure 3.6: Example: Mixed Reality Toolkit (MRTK3)

3.3.2 Key Features and Components

- **Unified Input & Interaction.** A comprehensive input system that abstracts device-specific interactions (e.g., hand tracking, eye gaze, voice commands) into a common interface. This allows developers to create consistent user experiences across multiple platforms without needing to write separate code for each device.
- **Extended UI Toolkit.** A set of pre-built UI components (buttons, sliders, menus) designed for MR environments. These components are optimized for spatial interactions and can be easily customized to fit the visual style of the application. The toolkit also includes a theming system for consistent branding across the UI.
- **Interactable Primitives & Simulation.** Ready-to-use prefabs for common MR interactions (e.g., near-far selection, object manipulation) paired with Unity Editor extensions and play-mode simulators. This combination allows rapid prototyping and testing of spatial interactions without requiring constant deployment to physical hardware.
- **Cross-Platform Abstractions & Extensibility.** A modular architecture that allows developers to extend or replace components as needed. This includes custom input providers, rendering pipelines, and interaction models. The extensibility framework is designed to be user-friendly, enabling developers to create and share custom components easily.
- **Performance Monitoring & Optimization.** Built-in profiling tools to monitor application performance, including frame rates, memory usage, and input latency. These tools help developers identify bottlenecks and optimize

their applications for the best possible user experience on HoloLens 2 and other devices.

For this thesis, MRTK3 was selected as the primary toolkit for building the HoloLens 2 application due to its comprehensive feature set, ease of use, and strong community support. The toolkit's modular architecture allows for rapid development and iteration, while its focus on cross-platform compatibility ensures that the application can be easily adapted for future devices and platforms. By leveraging MRTK3's capabilities, we aim to create a robust and user-friendly augmented reality experience for firefighters in emergency situations.

3.4 Server and Data Communication

3.4.1 Introduction

In this system it is crucial to maintain a continuous connection between the mobile app, the Kafka data broker, and the HoloLens 2 devices. The server acts as a bridge between these components, ensuring that data is transmitted in real time and that all devices are synchronized. This is particularly important in emergency situations where timely information can make a significant difference in decision-making and resource allocation.

Key responsibilities of the server include:

- **Real-Time GPS Data Transfer:** Receiving GPS data from the mobile app and relaying it to the hololens 2 device, ensuring that the location of each team member is accurately represented in the AR environment.
- **Kafka Integration:** Subscribing to relevant Kafka topics (e.g., firespread predictions, hazard alerts) via KafkaJS and relaying those messages to connected devices.
- **WebSocket Communication:** Establishing and maintaining WebSocket connections with the mobile app and HoloLens 2 devices, allowing for low-latency, bidirectional communication.

3.4.2 Server Overview

The server is implemented in Node.js, chosen for its event-driven, non-blocking I/O model that easily scales to handle multiple simultaneous WebSocket connections and Kafka consumers. Its core modules and workflow are as follows:

- **Express.js Host:** Provides HTTP routing for health checks and initial client handshakes, and upgrades selected endpoints to WebSocket.

- **WebSocket Manager:** Maintains a pool of client sockets (mobile and HoloLens 2), each identified by a unique device ID. Incoming GPS data from the mobile app is received, optionally transformed, and then forwarded to the corresponding HoloLens 2 socket.
- **KafkaJS Consumer:** Connects to one or more Kafka topics, consumes real-time event streams (e.g., flood model outputs, hazard notifications), and pushes those messages after minimal processing, to all subscribed HoloLens 2 clients via WebSocket.
- **Data Format and Parsing:** All messages are encoded as JSON objects for lightweight, language-agnostic transmission. GPS updates include latitude/longitude and timestamp fields; Kafka messages contain a topic identifier, payload data (e.g., radiative power, hazard type), and server-assigned metadata.

3.4.3 Data Flow and Communication Architecture

A high-level sequence of interactions is:

1. **Mobile App → Server:** The app opens a WebSocket and streams JSON-encoded GPS positions at regular intervals.
2. **Server → HoloLens 2:** Upon receipt, the server looks up the target device's socket and relays the GPS JSON payload immediately.
3. **Kafka → Server:** The KafkaJS consumer listens for new messages on configured topics, and upon message arrival, wraps them in a minimal JSON envelope.
4. **Server → HoloLens 2:** The server broadcasts or directs each Kafka-sourced alert to the appropriate HoloLens 2 clients.

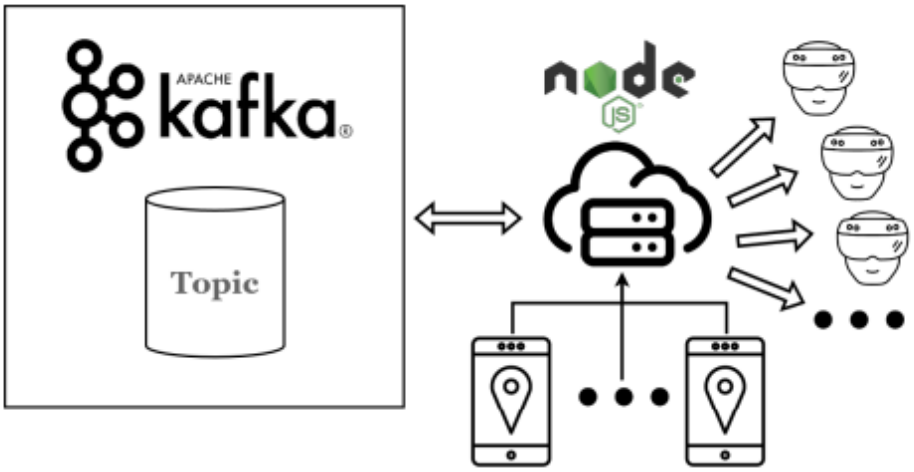

Figure 3.7: Server Communication Architecture

This architecture keeps the AR application lightweight deferring heavy event-streaming and protocol management to the server, while ensuring sub-second end-to-end latency for both location updates and external hazard data.

Chapter 4

Implementation

4.1 Introduction

This chapter focuses on the detailed implementation of the augmented reality (AR) wildfire emergency management System developed for Microsoft HoloLens 2. The System was created using the Unity Engine, with the Mixed Reality Toolkit 3 (MRTK3) enabling key AR interactions and user interface components. Its primary purpose is to enhance situational awareness for firefighting team captains by providing real-time visualizations of critical data directly within the user's field of view.

The implementation integrates multiple data sources, including live team positioning, environmental hazards, fire locations and fire-spread predictions, along with current and future weather conditions. These data are processed and displayed to the user through a combination of AR overlays, interactive maps, and AR markers.

The following sections will explore the key features of the system described below:

- **User Interface and Interaction Design:** Covers user-interface components such as menus, panels, and MRTK3-based interactions. Enables intuitive gesture and gaze for streamlined navigation and display of essential data.
- **Server Connection & Scene Calibration:** Establishes WebSocket Connection for data exchange, synchronizes GPS updates, and aligns AR content to real-world coordinates. Ensures accurate positioning and continuous, low-latency communication.
- **Map Integration and Routing:** Handles 3D map rendering with Mapbox, calculates optimal routes, and provides interactive path visualization. Supports rapid navigation decisions in emergency scenarios.

- **Team Members Locations & Points of Interest:** Shows real-time positions of team members and highlights POIs containing resources or potential safe and danger zones, both on the map and the AR scene. Enhances situational awareness and team coordination.
- **Active Fire Locations & Spread Forecast:** Retrieves information of active fire zones and visualize a fire-spread simulation based on environmental factors. Displaying dynamic hotspots for strategic planning.
- **Weather Information Overlay:** Integrates current and future weather data and presenting them to the user as AR overlays. Helps anticipate environmental changes and allocate resources.

Each section includes implementation details, representative code parts, and a discussion of technical challenges, optimizing the systems performance on HoloLens, and maintaining user safety under high-stress conditions, along with the strategies adopted to address them.

4.2 User Interface and Interactions

4.2.1 UI Panels and Layout

The user interface (UI) of the system is designed to be intuitive and responsive, allowing users to access critical information and controls with minimal cognitive load. The layout is structured around five main panels: the hand-following menu, the map panel, the weather information panel, and two system setup panels. Each panel serves a specific purpose and is designed to be easily accessible in the mixed-reality environment. The panels are arranged to provide a clear hierarchy of information, with the most important data and controls always available to the user.

The panels were designed using Unity's main building components, except for the hand-following menu, which was implemented using an MRTK3 prefab. The panels are visually distinct, with clear labels and icons indicating their functions. The layout is optimized for use in a mixed-reality environment, considering the user's field of view and the need for quick access to information.

Additionally, a custom script is attached to each panel to follow the user's field of view. The script uses Unity's Vector3.Lerp function to smoothly interpolate the panel's position toward a target location relative to the main camera, maintaining a consistent offset. This allows the panels to follow the user's gaze and remain in a fixed position within the mixed-reality environment. The script also includes functionality to hide and show the panels, with a toggle function that records the panel's current position relative to the camera and moves it off-screen when hidden. When the panel is shown again, it calculates a new position in front of the camera based on the camera's horizontal rotation (yaw) and the stored

distance, ensuring the panel reappears appropriately oriented toward the user before resuming its following behavior.

Hand-Following Menu

This menu is a redisigned menu based on the MRTK3 `HandConstraintPalmUp` prefab, which allows users to access various system functions and settings. The menu is designed to follow the user's hand movements, allowing the user to quickly access the functions they need without interrupting their workflow. The menu includes buttons to controll all the main system features, that are listed below:

- **Map Visibility.** Shows or hides the map panel, allowing users to focus on other information when needed.
- **Weather Panel.** Open the weather monitoring panel that current and future weather data.
- **Team Members Locations & POIs,** Displays the locations of team members and the POIs on the map and in the AR environment.
- **Fire Spread Simulations.** Shows or hides the fire spread simulation visualization on the map, allowing users to visualize potential fire spread patterns based on current conditions.
- **Active Fires.** Displays the locations of active fires on the map and on the AR environment, providing users with real-time information about fire activity in the area.
- **Destination Selection.** Allows users to select a destination on the map, which can be used for navigation and routing purposes.

Figure 4.1: Hand Following Menu Panel

Main Map Panel

The main map panel is a central component of the system, by displaying the user's and team members' locations, the POIs, and the active fire locations with 3D markers on the map, along with a route to the selected destination.

The map panel is designed to be interactive, allowing users to Unlock and adjust the map's position and scale, and place it in a convenient location in the AR environment, based on their preferences or the current situation needs. This is achieved by using the MRTK3 `ManipulationHandler` component, which enables users to manipulate game objects in the AR environment using hand gestures. Along with a custom script that recalculates the map's position and scale relative to the user when it's locked, ensuring that it follows the user's movements and remains in the selected new position on the AR environment.

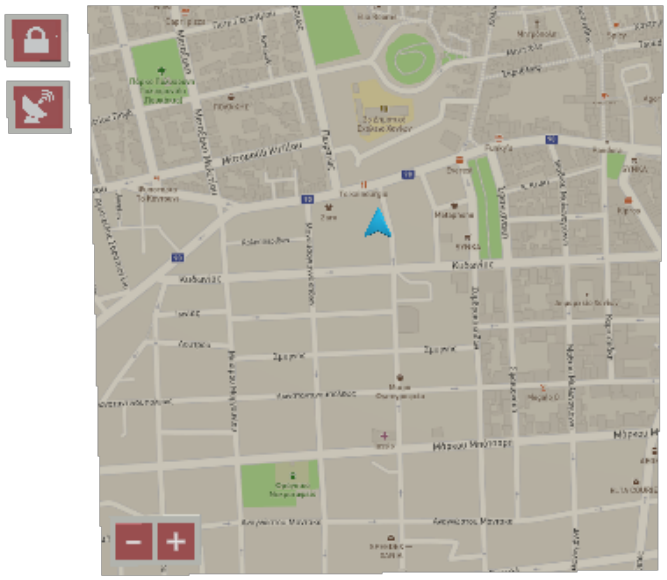

Figure 4.2: Main Map Panel

Weather Monitoring Panel

The weather monitoring panel is designed to display current and forecasted weather data, including temperature, wind speed, and wind direction. It also includes a toggle to show or hide 3D wind direction vectors on AR environment, and a slider to choose the forecasted time, allowing users to visualize how the weather conditions are expected to change over time. The panel is designed to be easy to read and understand, with clear labels and visual indicators for each data point.

Figure 4.3: Weather Monitoring Panel

System Set-Up Panels

The system setup panels are designed to allow users to configure the system settings, including the GPS location setup and device orientation calibration. These panels are designed to be user-friendly and intuitive, with clear instructions and visual cues to guide users through the system setup process. After the setup process is completed, the panels are hidden, and the system is ready for use. The Calibration panel can be accessed at any time to recalibrate the system if needed.

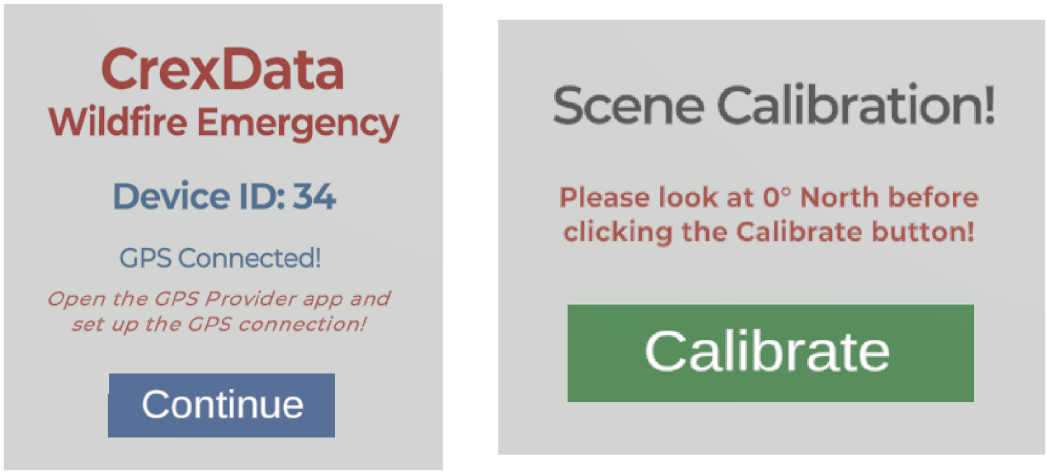

Figure 4.4: System Setup Panels

4.2.2 Interaction Methods

The system supports multiple interaction methods to accommodate different user preferences and scenarios provided by the MRTK3. These methods include:

- **Manual Interaction.** Users can interact with the system using hand gestures, such as tapping or dragging, to select and manipulate elements on the screen. This method is suitable for users who prefer direct manipulation of the interface and provides a tactile experience.
- **Distance Interaction.** Users can interact with the system from a distance using the hand ray pointer, which allows them to select and manipulate elements without needing to be in close proximity, with a ray casted from the user's hand. This method is useful for users who may be wearing gloves or have limited mobility, as it allows them to interact with the system without needing to reach out the target object.
- **Gaze Interaction.** The system supports gaze-based interaction, allowing users to select and interact with elements by looking at them and then confirming the selection with a hand pinch gesture. This method is particularly useful for users who may have difficulty using hand gestures or manual controls.

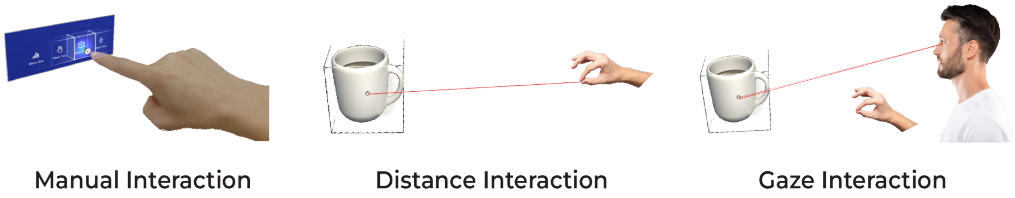

Figure 4.5: Different Interaction Methods

All the system panels and elements are designed to work with all the interaction methods, allowing users to choose the method that best suits their needs and preferences. The system also includes visual feedback to indicate when an element is selected or activated, providing users with clear confirmation of their actions.

4.3 Location Setup and Calibration Process

The location setup and calibration process is a critical step in ensuring the accuracy and reliability of the system's AR features. This process involves configuring the system to accurately represent the user's real-world location and orientation, to display the AR objects and information correctly in the user's field of view.

4.3.1 Problem Statement

The Hololens2 device does not provide a built-in method for setting the GPS location and orientation of the device, so the system does not have a way to determine the user's location and orientation relative to the real world. This can lead to inaccuracies in the AR experience, such as misaligned objects or incorrect positioning of information that is based on GPS data. To address this issue, the system includes a location setup and calibration process that allows users to correctly configure the system's GPS location and orientation. This process involves the following steps:

4.3.2 GPS Location Setup

To solve the problem of setting the GPS location, a custom phone application was developed. This application reads the GPS location of the user's phone, as a pair of latitude and longitude coordinates, and sends it to the server. The server then sends the GPS location to the HoloLens2 device, allowing the system to accurately determine the user's location in real time.

Phone & HoloLens pairing process The phone application and the HoloLens2 device are paired using a unique pairing ID, which is provided to the user during the setup process on the HoloLens2 device. The user enters this ID into the phone application, establishing a connection between the two devices. Once paired, the user can choose to start sharing their GPS location with the HoloLens2 device. Then the phone application continiusly sends the location of the user to the server using a WebSocket connection. The server then sends the GPS to the HoloLens2 device, allowing the system to accurately determine the user's location in real time.

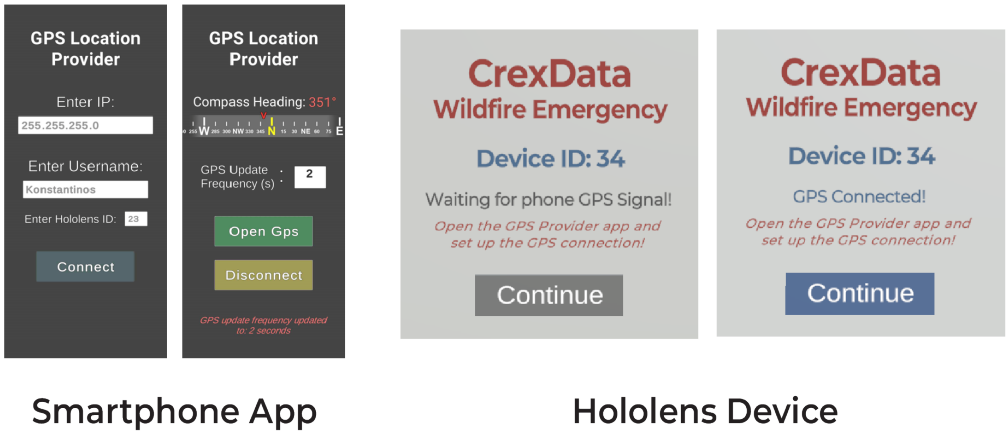

Figure 4.6: GPS Location Setup Process

4.3.3 Smartphone GPS Provider App

The phone application was also developed using Unity, and it is designed to run on Android devices. The application uses the Globalization library to access the GPS location of the user's phone, and when the user chooses to start sharing their location, the application calls the `SendGPSData()` Enumerator function, which conditiuslly reads the GPS location and sends it to the server as long as the Connection is active.

The phone application is designed to be user-friendly and easy to use, with a simple interface, allowing the user to quickly set up the pairing process and start sharing their GPS location.

```
IEnumerator SendGPSData()
{
    while (true)
    {
        if (isConnected && locationReady)
        {
            // Retrieve the phone's actual GPS data
            LocationInfo loc = Input.location.lastData;

            // Create JSON object
            var gpsMessage = new
            {
                type = "gps",
                senderId = loginHandler.hololensID,
                lat = loc.latitude,
                lon = loc.longitude,
                data = "Live GPS data from mobile device"
            };

            // Convert to JSON and send
            string jsonMessage = JsonConvert.SerializeObject(gpsMessage);
            SendMessageToServer(jsonMessage);

            Debug.Log($"Sent GPS: Lat {loc.latitude}, Long {loc.longitude}");
        }
        else
        {
            Debug.Log("WebSocket not connected or location services not running.");
        }

        yield return new WaitForSeconds(gpsSignalInterval);
    }
}
```

Figure 4.7: SendGPSData Function Code

4.3.4 Device Orientation Calibration

To solve the orientation problem, the system includes a calibration process that guides the user to align the the AR system's north with the real-world north. This process involves the following steps:

The user is prompted a guiding panel on the HoloLens2 device to look at the real-world north, guided by a compass inmplented in the phone application. The

compass conditiuslly shows where the phone is pointing, allowing the user to align his position with the real-world.

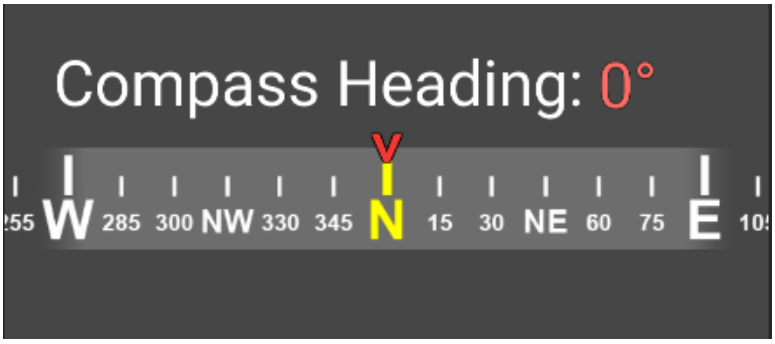

Figure 4.8: Compass in the Phone Application

After the user has aligned their position with the real-world north, they are prompted to press the calibration button on the AR panel. Then the system calls the CalibrateToNorth function, which calculates the angle between the real-world north and the AR system's north. This rotation angle is then applied to the cameraRoot object in the Unity scene, rotating the camera to align it correctly. The cameraRoot object is a parent object that contains the main camera and all other AR objects in the scene, ensuring that all objects are rotated correctly relative to the real-world north.

```
public void CalibrateToNorth()
{
    // Get the main camera's current forward vector
    Vector3 cameraForward = Camera.main.transform.forward;

    cameraForward.y = 0.0f;

    if (cameraForward.sqrMagnitude > 0.0001f)
    {
        cameraForward.Normalize();

        // Create a rotation that looks in the forward direction we just computed
        Quaternion targetRotation = Quaternion.LookRotation(cameraForward, Vector3.up);

        cameraRoot.rotation = targetRotation; // Apply this rotation to the root of the scene

        Debug.Log($"Camera north rotated to match the Real-World north");
    }
}
```

Figure 4.9: Calibration Function Code

Once the calibration process is complete, the system is ready to use, and the AR objects and information are displayed correctly in the user's field of view. The system also includes a recalibration button that allows the user to recalibrate the system at any time if needed.

4.4 3D Map (Mapbox)

This section describes the implementation of the 3D map using Mapbox, including the integration process, features, and functionalities. The Mapbox SDK was used to create a 3D map that displays the user's location, team members' locations, points of interest (POIs), and active fire locations. The map also includes a route calculation feature that allows users to navigate to a selected destination.

4.4.1 Mapbox SDK in Unity

The Mapbox SDK for Unity is a powerful tool that allows developers to create interactive maps and location-based applications using the Mapbox platform. The SDK provides a set of APIs and components that enable developers to easily integrate Mapbox maps into their Unity projects, allowing for the creation of custom maps, geolocation features, and real-time data visualization.

Integration Process

To integrate the Mapbox SDK into the Unity project, the following steps were followed:

- Download the Mapbox SDK for Unity from the Mapbox website.
- Import the SDK into the Unity project using the Unity Package Manager.
- Create a Mapbox account and obtain an access token to use the Mapbox services.
- Configure the Mapbox settings in Unity, setting the access token and map style.
- Create a new map object in the Unity scene and configure its properties, such as location, zoom level, and map style.

The Mapbox features implemented in the system are a 3D map display based on the user's GPS location and a route calculation feature that allows users to navigate to a selected destination.

4.4.2 Display the Map

To display the map in the AR enviroment, the user needs to select the map from the hand-following menu. Then the system calls the Mapbox API to generate the map tiles based on the user's GPS location and the selected zoom level, then the map is displayed in the AR environment, along with the user markerpointing to the user's GPS location.

Figure 4.10: Open the Map Panel

4.4.3 Additional Map Features

To have more interactive map, the system also includes three additional custom features that will be described below.

Map Repositioning

The map repositioning feature allows users to move the map in the AR environment by using hand gestures. To use this feature, there is a toggle lock/unlock button on the map panel. When the map is unlocked, the user can move the map by dragging it with their hand and place it the desired location in the AR environment based on the current situation or the user's preferences. When the map is locked again, a function is called to recalculate the map's offset position relative to the user's position, ensuring that the map follows the user and remains in the selected position in the AR environment.

Zoom In/Out

Additionally, the map includes a zoom in/out feature that allows users to adjust the map's scale to view a larger area or to zoom in and view more details around their location. This feature is implemented using a functions that changes the zoom variable in map script provided by Mapbox SDK and recalls the Mapbox API to update the map tiles based on the new zoom level. Also to keep the map marker in the correct position on the map after the scale change, a function is implemented that recalculates all the spawned markers' positions based on the new scale.

A problem that was encountered during the implementation of this feature was that the map generated by Mapbox SDK does not have a constant size when different zoom levels are selected. This means that the 3D map object size changes slightly when the zoom level is changed, which is causing overplaping problems between the map and the UI elements.

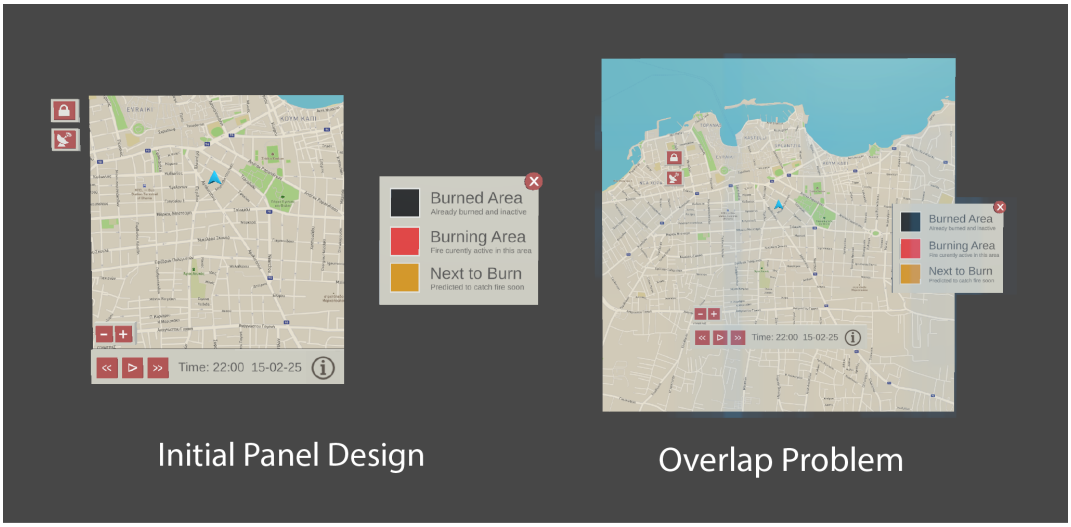

Figure 4.11: Map Overlapping Problem

To solve this problem, we used the MRTK3's Magic Windows prefab which is included in the MRTK3 package. The Magic Windows prefab consists of a square mesh with a transparent material that acts as a mask, and then there is another material that is applied to the objects that has to be masked.

To apply this technique to our problem, the square mesh is placed in front of the map with the desired size, and then added the second material to the map tiles and all the map markers. This way the system only renders elements that are inside the square mask, and keeps everything outiside the choosen area invisible for the user, maintaining a constant map size, preventing overlapping problems with the UI elements and providing a better experience for the user.

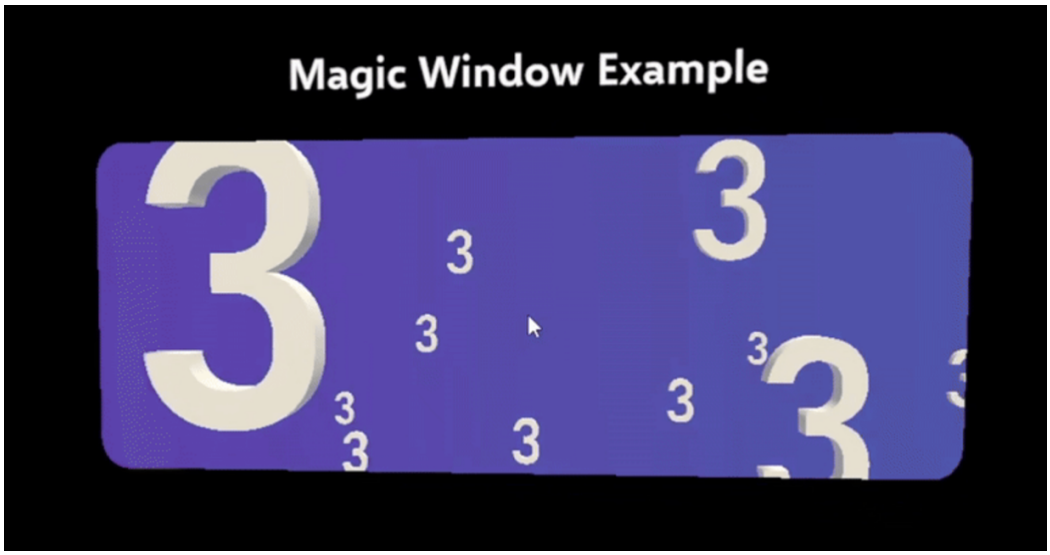

Figure 4.12: Magic Window Example

Street View and Satellite Mode

The map also includes a street view and satellite mode feature that allows users to switch between different map styles. This feature is implemented by a toggle button on the map panel, and a function that changes the selected style variable in the map script. The map styles are defined in the Mapbox settings, and the function recalls the Mapbox API to update the map tiles based on the selected style. This feature allows users to view the map in different perspectives, providing more context and information about their surroundings.

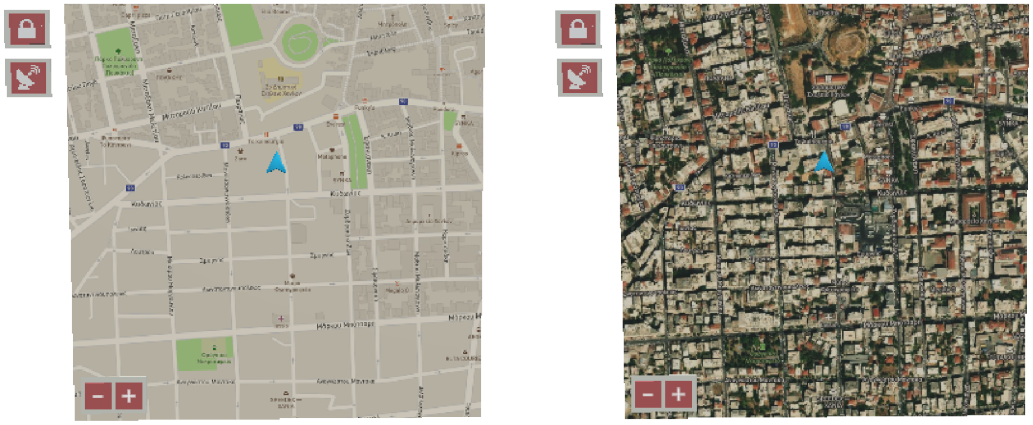

Figure 4.13: Street View and Satellite Mode

4.4.4 Spawn Markers on the Map

To spawn all the markers on the map such as the user marker, team members' markers, points of interest (POIs), and active fire locations in the correct position, the system uses a custom function called `DisplayOnMap`. This function takes six different input parameters explained below:

- **map:** The main map object where the markers will be displayed.
- **prefab:** The prefab of the current marker type that will be instantiated and displayed on the map.
- **GPSlocation:** The GPS coordinates of the marker location, passed as a `Vector2d` object.
- **scaleFactor:** The scale factor applied to the marker, depending on its type.
- **parent:** The parent object that will contain all markers of the same type, keeping the hierarchy organized.

- **name:** The name assigned to the marker object, useful for identification in the hierarchy and code.

The function instantiates the marker prefab and sets its parent to the specified parent object. It then translates the GPS coordinates to Unity world space using the `GeoToWorldPosition` function provided by the Mapbox SDK. After positioning the marker, it applies the specified `scaleFactor` to its local scale. The marker's rotation is then matched to that of the map to ensure alignment. Finally, the function assigns the given name to the instantiated marker object.

```
public static GameObject DisplayOnMap(AbstractMap map, GameObject prefab, Vector2d GPSlocation, float scaleFactor, Transform parent, string name){

      // Instantiate the map marker prefab
      GameObject mapMarker = GameObject.Instantiate(prefab);

      // Set the new object's parent
      mapMarker.transform.SetParent(parent);

      // Translate the GPS coordinates to Unity world position relative to the map
      mapMarker.transform.position = map.GeoToWorldPosition(GPSlocation, true);

      // Apply the zoom-adjusted scale to the marker
      mapMarker.transform.localScale = mapMarker.transform.localScale * scaleFactor;

      // Match the map's rotation
      mapMarker.transform.rotation = map.transform.rotation * Quaternion.Euler(90, 0, 0);

      mapMarker.name = name;

      return mapMarker;
}
```

Figure 4.14: Display Markers on Map Function

4.5 Navigation and Routing

This section describes the implementation of the navigation and routing features in the AR system. The routing functionality is designed to provide users with optimal routes to their selected destinations, such as points of interest (POIs), team members or selecetd locations on the map. The routing feature is implemented using the Mapbox Directions API, which provides real-time routing information based on the user's current location and the target destination point.

4.5.1 Route Calculation

To calculate the route, the system used a function that takes the user's current GPS latitude and longitude coordinates and the destination point coordinates as input parameters. The function then calls the Mapbox Directions API providing it with the starting and destination coordinates, and the mode of transportation (driving, walking, cycling, etc.), for our system we used the walking mode. Then the API gives a `respose` that cointains a list of GPS coordinates representing

all the checkpoints the user needs to follow between the starting point and the destination. The system then uses these coordinates to create a line renderer object that connects all the checkpoints, displaying the route on the map.

4.5.2 Route Visualization

After the route is calculated, the system procceds to visualize the route on the maplong with an AR Guidance marker that shows the direction to follow.

Figure 4.15: Line Renderer Setup Function

3D Map Display

To display the route on the map, the system uses the Unity's Line Renderer component, which creates a line connecting all the given points in the unity world space. In our case to correctly display the route on the map, the system first converts the GPS coordinates of the route checkpoints to Unity world space using the `GeoToWorldPosition` function provided that was explained in the previous section.

Figure 4.16: Traslating GPS Coordinates to Unity World Space

The system then calls a new function called `LineRendererSetup` that takes the new route coordinates and creates a new game object with the Line Renderer component. The function also sets the material, width, and other properties of the line renderer to customize its appearance and placing it correctly on the map, to match the system's design and provide a clear visual representation of the route.

```
public static GameObject LineRendererSetup(GameObject _routeGO, AbstractMap map, List<Vector3> routeGeometry, Material routeMat, float routeWidth){

        _routeGO = new GameObject("Route");
        _routeGO.transform.SetParent(map.gameObject.transform.parent);

        List<Vector3> positions = routeGeometry.ToList();

        // Convert world positions to local positions relative to _routeGO
        for (int i = 0; i < positions.Count; i++)
        {
            positions[i] = _routeGO.transform.InverseTransformPoint(positions[i]);
        }

        // Add and configure LineRenderer
        LineRenderer lineRenderer = _routeGO.AddComponent<LineRenderer>();
        lineRenderer.positionCount = routeGeometry.Count;
        lineRenderer.SetPositions(positions.ToArray());
        lineRenderer.material = routeMat;
        lineRenderer.widthMultiplier = routeWidth;
        lineRenderer.useWorldSpace = false;
        lineRenderer.alignment = LineAlignment.View;

        // Apply a small offset to the Z position of each point
        Vector3[] positionsTMP = new Vector3[lineRenderer.positionCount];
        lineRenderer.GetPositions(positionsTMP);

        for (int i = 0; i < positionsTMP.Length; i++)
        {
            positionsTMP[i].z = positionsTMP[i].z - 0.0001f;
        }

        // Update the LineRenderer with the modified positions
        lineRenderer.SetPositions(positionsTMP);

        // Optional visual enhancements
        lineRenderer.sortingLayerName = "Default";
        lineRenderer.sortingOrder = 0;

        return _routeGO;
    }
```

Figure 4.17: Line Renderer Setup Function

AR Route Guidance

To provide users with real-time guidance along the route, the system also includes an AR marker that points to the next checkpoint in the real world. This marker

is designed to be easily visible and provides users with a clear indication of the direction they need to follow. The AR marker is a 3D arrow model designed in blender, and it is placed in the AR environment at a fixed distance in front of the user, following his movements.

For the arrow to correctly point to the next checkpoint, the system must know the next checkpoint location in scene coordinates. To achieve this, the system uses a function called `CalculatePositionInScene` that takes the user's GPS coordinates and the next checkpoint GPS coordinates as input parameters. The function then calculates the position of the next checkpoint in the Unity world space by applying a geospatial conversion formula. The function uses the WGS84 ellipsoid model to convert the latitude and longitude coordinates to meters. The formula used in the function is based on the Haversine formula, which calculates the distance between two points on a sphere given their latitude and longitude coordinates.

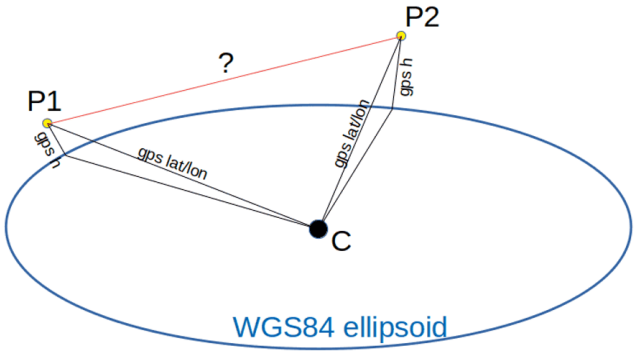

Figure 4.18: WSG84 Ellipsoid Model

The resulting Unity coordinates is then used to place an invisible marker at the next target location in the AR scene. Then the system calls the `UpdateArrow` function that calculates the direction vector from the user's position to the marker's position, and uses this vector to rotate the 3D arrow marker directly towards the next checkpoint. The arrow then is placed at a fixed distance in front of the user slightly above the ground, and it follows the user's movements in the AR environment.

```
public static void UpdateArrow(RoutePlanner rp)
{
    // Check if there is a next waypoint to navigate to
    if(rp.waypointScenePositions.Count > rp._nextWaypointIndex){

        // Get the next waypoint(checkpoint) position and user position
        Vector3 nextWaypoint = rp.waypointScenePositions[rp._nextWaypointIndex];
        Vector3 userScenePosition = Camera.main.transform.position;
        Vector3 direction = nextWaypoint - userScenePosition;

        // Calculate the distance to the next waypoint
        Vector2 userScenePosition2D = new Vector2(userScenePosition.x, userScenePosition.z);
        Vector2 nextWaypointPos2D = new Vector2(nextWaypoint.x, nextWaypoint.z);
        float distanceToWaypoint = Vector2.Distance(userScenePosition2D, nextWaypointPos2D);

        float waypointReachThreshold = 2.0f;

        // Update next waypoint index if user has passed the current one
        if (distanceToWaypoint < waypointReachThreshold)
            rp.ShowRoute(rp._endLocation);

        // Calculate the arrow's new position, a fixed distance from the base object
        rp._arrowInstance.transform.position = userScenePosition + direction.normalized * 2.0f + new Vector3(0, -1.6f, 0);

        // Rotate the arrow to point toward the next waypoint
        Vector3 arrowRotationOffset = new Vector3(90, 180, 0);
        float angle = Mathf.Atan2(direction.z, direction.x) * Mathf.Rad2Deg;
        Vector3 arrowRotation = new Vector3(0, -angle, 0) + arrowRotationOffset;

        if (direction != Vector3.zero)
            rp._arrowInstance.transform.eulerAngles = arrowRotation;
    }

}
```

Figure 4.19: Update Arrow Function

4.6 Team Members' Locations Visualization

This section describes the implementation of the team members' locations visualization feature in the system. This feature is designed to provide users with real-time information about the locations of their team members, enhancing situational awareness and coordination during firefighting operations.

4.6.1 Phone - Server - HoloLens Communication

Provide the GPS data to the server

To provide the GPS data to the server, each team member must have an android phone device with the GPS location provider application installed, the same application that was used to set the GPS location of the user. The application then continuously reads the GPS location of the user's phone and sends it to the server using a WebSocket connection, along with the user's name.

Read the GPS data from the server

After the server receives any GPS data from from the team members, it instantly sends the data in JSON format to the team opperator's hololens device, tagged with the "teamMember" message type to identify the data. The system then reads the server's message and extracts the important information such as the user's name, id, and GPS latitude and longitude coordinates. The system then

calls the `UpdateFirefighterPosition` function checks if the user is already in the list of team members based on his id. If the user is not in the list, it it calls the `AddFirefighter` to create a new firefighter object and add it to the list. If the user is already in the list, it just updates the user's GPS coordinates.

```
public void UpdateFirefighterPosition(int id, string name, Vector2d gpsLocation){
    // Find the firefighter by ID
    Firefighter firefighter = firefighters.Find(f => f.id == id);
    if (firefighter != null)
    {
        // Update the firefighter's location
        firefighter.GPSlocation = gpsLocation;
    }
    else
    {
        AddFirefighter(id, name, gpsLocation);
        Debug.LogWarning($"Firefighter with ID {id} not found. Adding new firefighter.");
    }
}

void AddFirefighter(int id, string name, Vector2d gpsLocation)
{
    Firefighter firefighter = new Firefighter { id = id, Name = name, GPSlocation = gpsLocation };
    firefighters.Add(firefighter);
}
```

Figure 4.20: Update Firefighter Position Function

4.6.2 Display Team Members' Locations

After the sysyem stores the GPS data of the team members, it can display their locations both in the AR environment and on the map.

Display team members as markers on the Map

To display the team members on the map, the system uses the same `DisplayOnMap` function that was explained in the previous section. The system calls this function for each team member in the list, passing the user's GPS coordinates and the corresponding marker prefab. The function then instantiates a new marker object for each team member and places it on the map at the corresponding position.

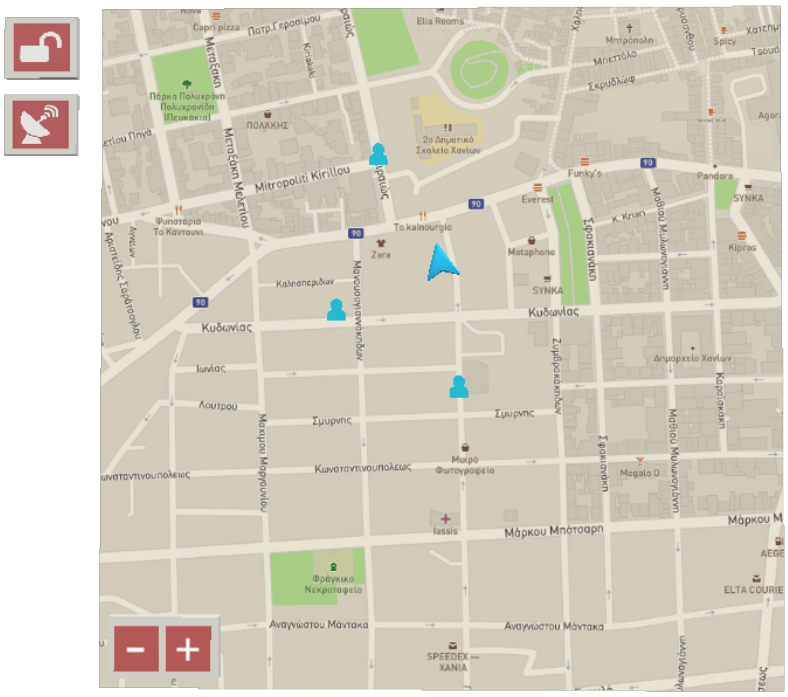

Figure 4.21: Team Members Display on the Map

Placing 3D overlays in the AR Environment

To display the team members in the AR environment, the systemfirst translates the GPS coordinates of each team member to Unity world space using the same technique that was used to place the checkpoints in the scene for the routing feature. And the it spawns a 3D human figure prefab at the translated coordinates. The human figure helps the user to easily identify the positions of the team members in the AR environment, and it is easily visible and distinguishable from other AR objects.

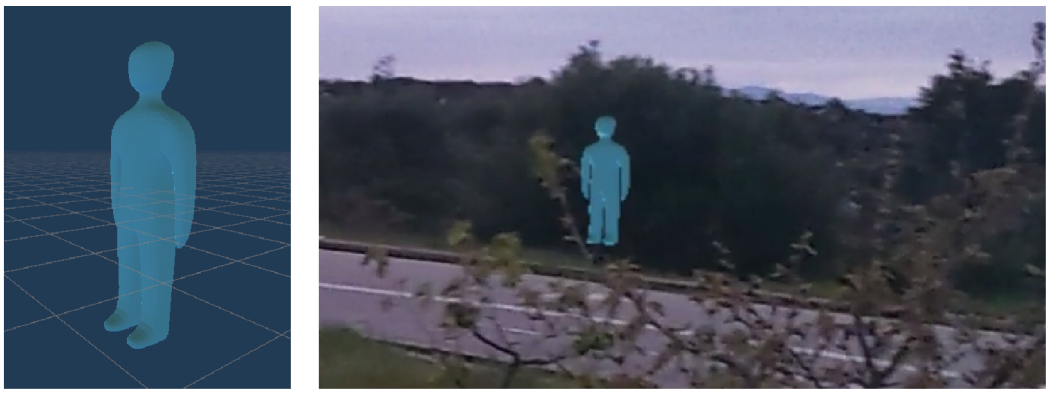

Figure 4.22: 3D Human Figure Prefab

Navigating to a Team Member

To navigate to a team member, the user can click on any team member's marker on the map, which will open a pop-up menu with the team member's name and a button to navigate to their location. When the user clicks the `Show Route` button, which calls the `CalculateRoute` function from the RoutePlanner script, passing the selected team member's GPS coordinates as the destination point. The system then calculates the route to the selected team member and displays it on the map, along with the AR guidance marker pointing to their location. This feature allows user to quickly and easily navigate to their team members locations, enhancing coordination and communication during firefighting operations.

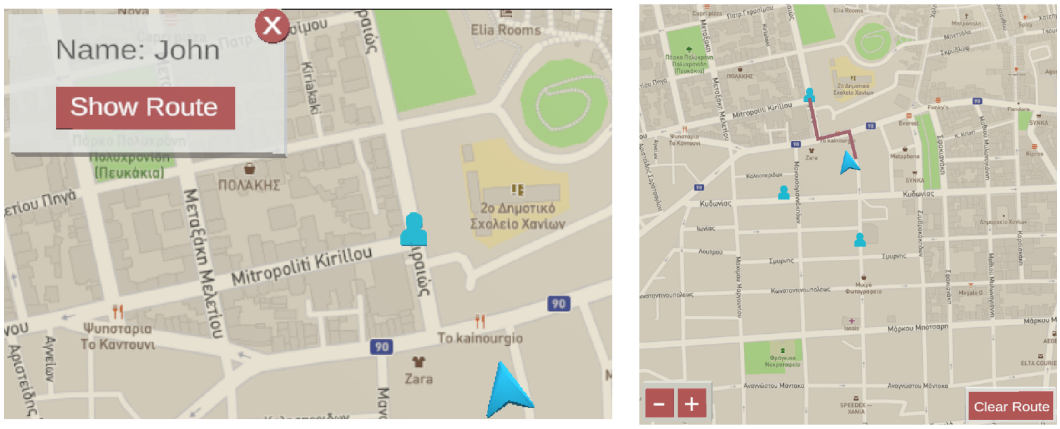

Figure 4.23: Navigating to a Team Member

4.7 Points of Interest (POIs) Visualization

This section describes the implementation of the points of interest (POIs) visualization feature in the system. The POIs are critical locations or resources that are relevant to firefighting operations. The system is designed to display these POIs in both the AR environment and on the map, providing users with real-time information about their surroundings.

4.7.1 POI Design and Structure

The POIs are designed to be easily identifiable and distinguishable from other AR objects in the scene. Currently, the system is designed to be extensible, allowing for the addition of new POI types in the future, but currently it includes the following types of POIs:

- **Water sources.** Indicating the location of water sources in the area.
- **Power Lines.** Pointing to the locations of power lines in the area, which can be hazardous during firefighting operations.
- **Danger Zones.** Represented by a danger symbol, indicating spots that are hazardous or unsafe for firefighting operations.
- **Settlement Areas.** Indicating the locations of settlements or populated areas close to the fire zone.

Each POI type is represented by a unique vector image, designed to be easily recognizable and distinguishable from other POIs.

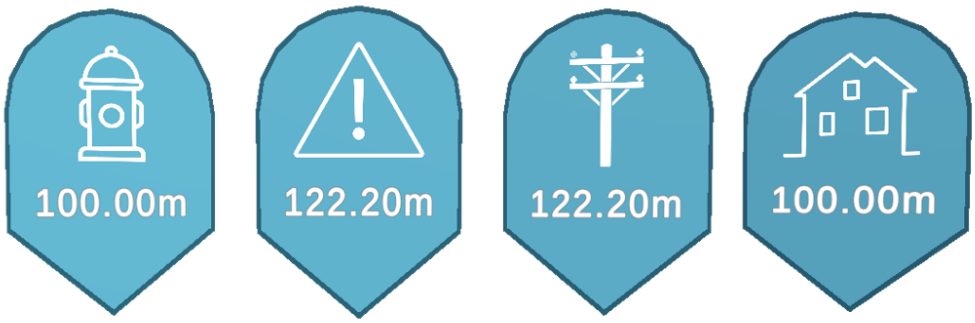

Figure 4.24: Types of POIs

4.7.2 Receiving POI Data from the Server

The system receives the POI data from the server in JSON format, tagged with the "spawnPOI" message type to identify the data, similar to the team members' locations data. The server sends the POI data to the HoloLens2 device, which then reads the message and extracts the name, the type, and the GPS coordinates of the POI. The system then calls the `AddPOI` function given the POI data, to create a new POI object and add it to the list of POIs, to be displayed later in the AR environment and on the map.

```
public void AddPOI(string name, double latitude, double longitude, string type)
{
    POI poi = new POI { Name = name, Latitude = latitude, Longitude = longitude, type = ChoosePOIType(type) };
    pointsOfInterest.Add(poi);
}

private POI_Type ChoosePOIType(string type)
{
    switch (type)
    {
        case "WaterSource":
            return POI_Type.WaterSource;
        case "SettlementArea":
            return POI_Type.SettlementArea;
        case "PowerLine":
            return POI_Type.PowerLine;
        case "RiskZone":
            return POI_Type.RiskZone;
        default:
            Debug.LogError("Invalid POI type: " + type);
            return POI_Type.RiskZone; // Default case
    }
}
```

Figure 4.25: Add POI Function

4.7.3 POI Display on the Map

The system displays each Point of Interest on the map using the `SpawnPOIOnMap` function, which sets a scale factor and then calls the `DisplayOnMap` along with the appropriate input parameters, similar to the team members' display. The funtion then spawns a new POI marker on the map at the corresponding GPS coordinates.

```
private void SpawnPOIOnMap(POI poi)
{
    if (map == null)
    {
        Debug.LogWarning("Map or mapMarkerPrefab not assigned, cannot spawn POI on map.");
        return;
    }

    // Convert the POI's latitude/longitude to a Unity world position on the map
    Vector2d geoPosition = new Vector2d(poi.Latitude, poi.Longitude);

    // Adjust the base scale factor based on the zoom level
    float scaleFactor = 0.025f;

    // Instantiate the marker on the mapParent(POI_Parent.transform);
    GameObject mapMarker = ExtraFunctionsMapbox.DisplayOnMap(map, GetPoiTypePrefab(poi.type, 1), geoPosition, scaleFactor, POI_MapParent.transform, "Map_" + poi.Name);
    POIMapObject poiMap = mapMarker.GetComponent<POIMapObject>();
    poiMap.GPSLocation = geoPosition;
    poiMap.routePlanner = routePlanner;

    // Track this marker for later removal
    spawnedMapMarkers.Add(mapMarker);
}
```

Figure 4.26: Spawn POI Function

POI Selection and Navigation

After the POIs are displayed on the map, the user can click on any POI marker that he want to navigate to, after a POI is selected, the system calls the `ShowRoute` function from the RoutePlanner script, along with the selected POI GPS coordinates as the destination point. The system then calculates the route to the selected POI and displays it on the map, providing the user with a path to critical locations or resources. The AR guidance marker is also updated to point to the selected POI, allowing the user to navigate easily to the POI in the real world.

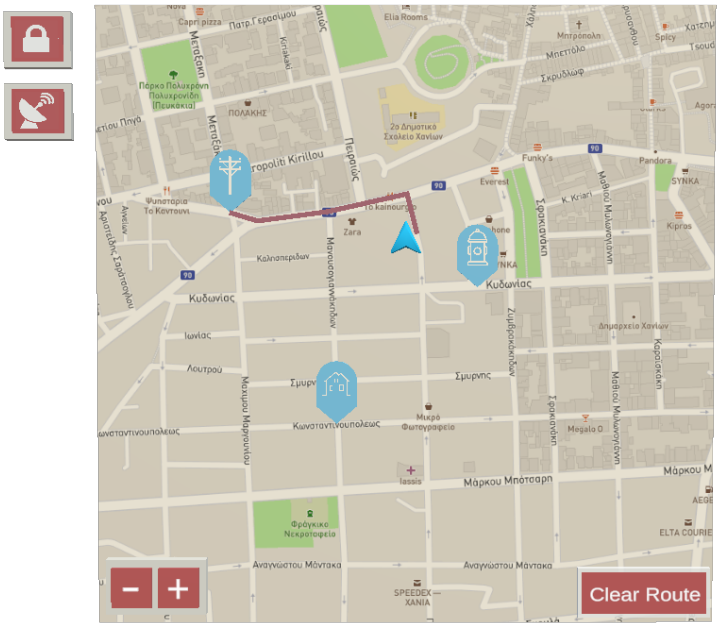

Figure 4.27: POI Display on the Map

4.7.4 POI Display in the AR Environment

The system also displays the POIs in the AR environment, providing users with real-time information about their surroundings. The system uses the same technique that was used to display the team members in the AR environment, translating the GPS coordinates of each POI to Unity world space using the `CalculatePositionInScene` function. Then spawing the POI prefab at the translated coordinates, along with some additional settings for the objects parrent and name in the hierarchy, to keep the scene organized.

Figure 4.28: POI Display in the AR Environment

POI Distance Indicator

To provide users with additional information about the POIs, each POI prefab includes a distance indicator that shows the distance from the user to the POI. The distance indicator is a simple text label that displays the distance in meters, and it is updated every few seconds to reflect the user's current position. The distance is calculated by subtracting the user's position from the POI position in the Unity world space, using the `Vector3.Distance` unity function.

```
void Update()
{
    // Calculate distance from the user to this object
    float distance = Vector3.Distance(Camera.main.transform.position, transform.position);

    distanceTimer += Time.deltaTime;
    if(distanceTimer > textUpdateRate){

        updateText(distance);
        distanceTimer = 0;
    }
}
```

Figure 4.29: POI Distance Update Code

Visibility Issues and Solutions

When displaying the POIs in the AR environment, the system encountered visibility issues with the POIs that were far away from the user. The POIs were not clearly visible in the AR environment, making it difficult for the user to identify their type and their distance. To solve this problem, the system includes a custom script attached to each POI prefab on the scene, which based on the distance from the user that was calculated in the previous section, adjust the prefab's scale to be larger when the user is far away from the POI, and smaller when the user is close to the POI. This way, the system ensures that all POIs are clearly visible in the AR environment, regardless of their distance from the user.

```
void Update()
{

    // Calculate distance from the user to this object
    float distance = Vector3.Distance(Camera.main.transform.position, transform.position);

    scaleTimer += Time.deltaTime;
    if(scaleTimer > sizeUpdateRate){

        // Clamp the distance to avoid zero or extremely large scale
        distance = Mathf.Clamp(distance, minScaleDistance, maxScaleDistance);

        float scale = sizeAtOneMeter * distance/2;
        float distanceSizeReducer = distance/7;

        // Apply the scale to the transform uniformly
        transform.localScale = (Vector3.one * (scale - (scale/100)*distanceSizeReducer));
        transform.localPosition = new Vector3(transform.localPosition.x, (distance/8.5f), transform.localPosition.z);

        scaleTimer = 0;
    }
}
```

Figure 4.30: POI Scale Adjustment Code

Each POI prefab also includes a script that rotates the prefab to always face the user, ensuring that the POIs are always visible from every position in the AR scene. This is achieved by using the `LookAt` function in Unity, which rotates the prefab to face a target object in the unity world. In this case, the target object is the user's camera, which resemples the hololens position in the AR environment.

4.8 Fire Monitoring

This section describes the implementation of the fire monitoring feature in the system. The fire monitoring feature is designed to provide users with real-time information about active fires in the area, helping them to make informed decisions during firefighting operations. The system is designed to display the active fire locations in both the AR environment and on the map, providing users with real-time information about the fire's location and behavior.

4.8.1 Active Fire Locations

The system receives the active fire locations data from the NASA FIRMS API, which provides information about active fires detected by satellite, three hours after the fire is detected. The Data is provided to the system in JSON format, each fire location is represented by a latitude and longitude coordinates, along with additional information such as the fire's confidence level and brightness. The system then parses the JSON file to extract the relevant information and store it in a list of fire data objects, which are then proccessed and displayed in the AR environment and on the map.

Example of the JSON response from the NASA FIRMS API:


```
{
    "latitude": 38.37386,
    "longitude": 24.06201,
    "acq_date": "2025-05-12",
    "acq_time": "0005",
    "confidence": "n",
    "brightness": 300.65,
    "satellite": "N21",
    "track": 0.38,
    "instrument": "VIIRS",
    "bright_t31": 284.73,
    "frp": 0.97,
    "version": "2.0NRT",
    "scan": 0.42,
    "daynight": "N"
}
```


Figure 4.31: NASA FIRMS API JSON Response

The JSON response contains a list of fire data objects, each containing the following information:

- **latitude & longitude.** The GPS coordinates of the fire location, indicating its position in the world.
- **brightness.** The brightness value of the fire, indicating its intensity.
- **confidence.** The confidence level of the fire detection, indicating the reliability of the data.
- **acq_date, acq_time & daynight.** The date and time of the fire detection, along with the day or night indicator, indicating whether the data was collected during the day or night.

- **satellite, instrument & version.** The satellite and instrument used to detect the fire, along with the version of the data.
- **bright_t31 & frp.** Additional data related to the fire detection, such as the brightness temperature and fire radiative power.
- **scan.** The scan angle of the satellite when the fire was detected.

Then used a python script to parse the JSON response and erase all the unnecessary data, and focus only on a certain area based on the user's location. The script then saves the filtered data in a new JSON file, which is then used by the Unity system to display the active fire locations.

Display Fire Locations on Map

To display the active fire locations on the map, the system first has to Deserialize the JSON file to extract the fire data objects. To achieve this, the system uses a custom class called `FireData` that contains all the properties of the fire data objects.

```
public class FireData
{
    public float latitude { get; set; }
    public float longitude { get; set; }
    public string acq_date { get; set; }
    public string acq_time { get; set; }
    public float brightness { get; set; }
    public float frp { get; set; }
}
```

Figure 4.32: FireData Class

The system then uses the `JsonUtility` class provided by Unity to deserialize the JSON file and create a list of fire data objects. Then when the user choose to display the fire locations on the map, the system calls the `DisplayFireLocations` function, which takes the list of fire data objects that the system created to spawn a new marker at the fire's location on the map for each one of the active fires on the list. To display the fire locations on the map, the system uses the same `DisplayOnMap` function that was explained in the previous sections, passing the fire data objects as input parameters. The function then instantiates a new fire marker object and places it on the map at the corresponding position.

```
// Load the fire data from a JSON string
public List<FireData> LoadFireData(string json)
{
    return JsonConvert.DeserializeObject<List<FireData>>(json);
}

// Display the fire locations on the map
public void DisplayFireLocations(List<FireData> fireDataList)
{
    // Adjust the scale factor based on the current zoom level
    float currentZoom = map.Zoom;
    float baseScaleFactor = 0.1f * ScaleAdjustment * currentMapZoomLevel;

    // Display each fire location on the map
    foreach (var fire in fireDataList)
    {
        Vector2d fireLocation = new Vector2d(fire.latitude, fire.longitude);
        GameObject marker = ExtraFuncMapbox.DisplayOnMap(map, fireMarkerPrefab, fireLocation, baseScaleFactor, mapParent.transform, "fire_marker");

        fireMarkersList.Add(marker);
        SpawnPOIInScene(fireLocation);
    }
}
```

Figure 4.33: Parse JSON File & Display Fire Locations Functions

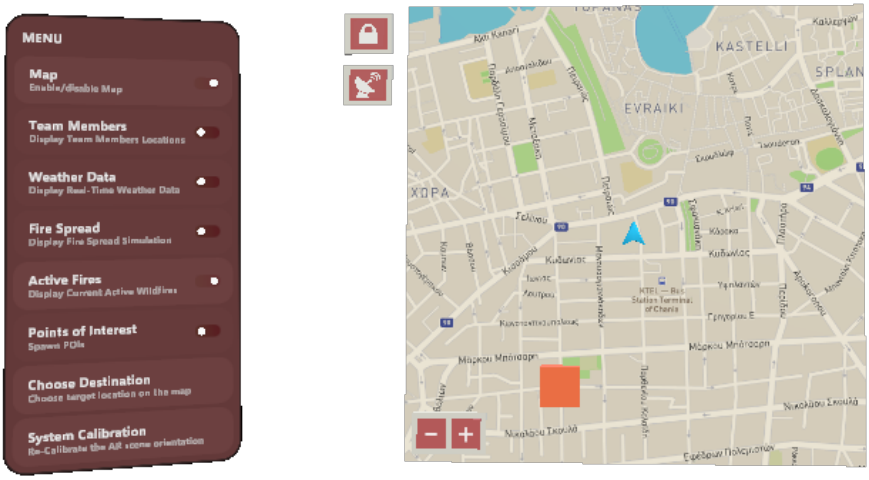

Figure 4.34: Active Fire Locations Display on the Map

3D Representation in AR

To display the active fire locations in the AR environment, the system uses again same technique that was used to display the team members and POIs in the AR environment. The system first translates the GPS coordinates of each fire location to Unity world space using the `CalculatePositionInScene` function. Then spawns the 3D fire prefab at the translated coordinates.

The fire prefab is a 3D model designed to represent the fire in the AR environment, similar to the POIs prefabs, and it also includes the distance indicator that shows the distance from the user to the fire location. The distance is calculated using the same technique that was used for the POIs, and it is updated every few seconds to reflect the user's current distance from the fire.

Figure 4.35: Active Fire Locations Display in the AR Environment

4.8.2 Fire Spread Visualization

The fire spread visualization feature is designed to provide users with real-time information about the potential spread of the fire in the area. The system uses the output data provided by the SPARK fire spread model to calculate the potential spread of the fire based on the current weather conditions and the fire's location.

SPARK's output data

The SPARK fire spread model provides a set of parameters that describe the potential spread of the fire, including the fire's location, arival time, the fire's intensity and flame height. These data are aranged as a grid of cells, each cell representing a specific 30m x 30m area in the world and containing the fire data for that area the time of the fire's arrival.

These data are provided in a CSV file format, that contains the following columns:

- **X.** The X coordinate of the cell center in Web Mercator (EPSG:3857) projection.
- **Y.** The Y coordinate of the cell center in Web Mercator (EPSG:3857) projection.
- **Arrival Time.** The time of the fire's arrival in the cell, in seconds since the start of the simulation.
- **Flame Height.** The height of the flames in the cell, in meters.
- **Fire Intensity.** The intensity of the fire in the cell, in kilowatts per meter (kW/m).

Visualization on the Map

The system then uses a function called `ParseFireSpreadData` to read the CSV file and extract the relevant data. It splits the CSV file into rows and then creates a new theoretical grid of cells, each cell representing a cell of the fire spread data. This grid is actualy an object list of a custom class called `GridCellObject` that contains the X and Y coordinates of the cell in the theoretical grid, along with the fire data for that cell. The function then instantiates all these objects on the scene, with the corresponding color indicating their current state, based on the fire's arrival time.

There are three different colors used to represent the fire spread data:

- **Red.** The cells where the fire is at the moment.
- **Yellow.** Indicates that the fire will arrive in the cell in the next 5 minutes.
- **Black.** The burned area, indicating that the fire has already passed through the cell.

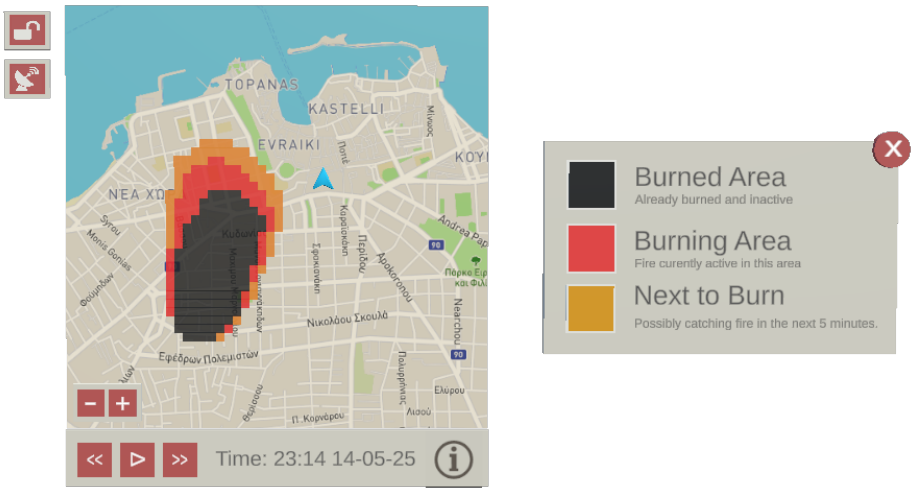

Figure 4.36: Fire Spread visualization on the Map

Controlling the Fire Spread Visualization

The system uncludes a simple UI panel that allows the user to control the fire spread visualization. The panel includes two back and forward buttons that allow the user to navigate through the fire spread data, with a time interval of 1 minute, and a play/pause button that allows the user to start and stop the simulation.

When the user clicks the forward button, the system increases the time variable by 1 and then calls the `UpdateVisualization` function with the current timestep,

this function parses through the list of grid cells and updates the color of each cell based on the fire's arrival time, to reflect the current state of the fire spread.

```
public void UpdateVisualization(int timeStep){

    foreach (var cell in grid)
    {
        if(cell.arrivalTime > timeStep - 300 && cell.arrivalTime <= timeStep)
            cell.UpdateCell(burningMAT);
        else if (cell.arrivalTime > timeStep && cell.arrivalTime <= timeStep + 300)
            cell.UpdateCell(willburnMAT);
        else if (cell.arrivalTime <= timeStep - 300)
            cell.UpdateCell(burnedMAT);
        else
            cell.UpdateCell(nothingMAT);
    }
}
```

Figure 4.37: Update Fire Spread Visualization Function

The same process is followed when the user clicks the back button, but in this case the time variable is decreased by 1. The play/pause button works in a similar way, by calling a coroutine that updates the fire spread visualization every second, until the user clicks the button again to stop the simulation. To indicate the time of the state of the fire that the user is currently viewing, the system includes a text label with the time and date of the data being displayed. The label is updated every time the state of the fire is updated.

Along with the controll buttons there is also an information panel that describes the meaning of each color used to represent the fire states to the user. The panel is designed to be easy to use and intuitive, allowing the user to quickly navigate through the fire spread data and understand the potential spread of the fire in the area, helping them to make informed decisions during firefighting operations.

4.9 Weather Data Monitoring

This section describes the implementation of the weather data monitoring feature in the system. The access to real-time weather data is crucial for firefighting operations, as it provides users with important information about the current and forecasted weather conditions in the area, helping them to make more accurate predictions of the fire behavior and progress in the area.

4.9.1 Current and Forecasted Weather

The main weather panel is based on a panel prefab of the MRTK3 package, and was modified to fit the system's design, and to display the weather data in a clear and organized way. The weather panel includes the following information:

- **Current Temperature.** Displayed in Celsius degrees.
- **Wind Speed.** Displayed in meters per second (m/s).
- **Wind Direction.** Displayed in degrees, indicating the direction from which the wind is blowing.

Along with an uncertainty value for each one of the parameters, indicating the level of uncertainty in the data.

Figure 4.38: Weather Data Panel

Data Source

At the moment, the systems is receiving the weather data from the OpenMeteo API, which provides real-time weather data and predictions for the next 24 hours, with 1 hour intervals, for any location in the world. The system calls the Open-Meteo API through the `StreamWeatherData` function, which sets the latitude and longitude coordinates of the user's location to the API request URL, and then calls the API using the `UnityWebRequest` class. The function then waits for the API response, and when the data is received, it parses the JSON response through the `ProcessWeatherData` function to extract the relevant weather information.

```
IEnumerator StreamWeatherData()
{
    double lat = targetLocation.x;
    double lon = targetLocation.y;

    Debug.Log($"Fetching weather data for: {lat}, {lon}");

    string url = string.Format(weatherAPIUrl, lat, lon);
    using (UnityWebRequest request = UnityWebRequest.Get(url))
    {
        yield return request.SendWebRequest();

        if (request.result != UnityWebRequest.Result.Success)
        {
            Debug.LogError($"Error: {request.error}");
        }
        else if (weatherDataManager != null)
        {
            Debug.Log(request.downloadHandler.text);
            weatherDataManager.ProcessWeatherData(request.downloadHandler.text);
        }
    }
}
```

Figure 4.39: Request Weather Data Function

The API request URL used in the system is:
`https://api.open-meteo.com/v1/forecast?latitude={0}&longitude={1}&current_weather=true&hourly=temperature_2m,windspeed_10m,winddirection_10m&forecast_days=1`

The `ProcessWeatherData` function then splits the JSON response to extract the current temperature, wind speed, and wind direction values, and stores them in a list of weather data objects to be later displayed to the user.

The JSON response from the OpenMeteo API is structured as follows:


```
{"latitude":35.5,"longitude":24.0,"generationtime_ms":0.054001808166503906,"utc_offset_seconds":0,"timezone":"GMT","timezone_abbreviation":"GMT","elevation":12.0,

"current_weather_units":{"time":"iso8601","interval":"seconds","temperature":"°C","windspeed":"km/h","winddirection":"°","is_day":"","weathercode":"wmo code"},
"current_weather":{"time":"2025-05-11T02:15","interval":900,"temperature":17.1,"windspeed":5.0,"winddirection":111,"is_day":0,"weathercode":0},

"hourly_units":{"time":"iso8601","temperature_2m":"°C","windspeed_10m":"km/h","winddirection_10m":"°"},
"hourly":{"time":["2025-05-11T00:00","2025-05-11T01:00","2025-05-11T02:00","2025-05-11T03:00","2025-05-11T04:00","2025-05-11T05:00","2025-05-11T06:00","2025-05-11T07:00","2025-05-11T08:00","2025-05-11T09:00","2025-05-11T10:00","2025-05-11T11:00","2025-05-11T12:00","2025-05-11T13:00","2025-05-11T14:00","2025-05-11T15:00","2025-05-11T16:00","2025-05-11T17:00","2025-05-11T18:00","2025-05-11T19:00","2025-05-11T20:00","2025-05-11T21:00","2025-05-11T22:00","2025-05-11T23:00"],
"temperature_2m":[17.6,17.4,17.2,16.8,16.8,18.0,19.1,19.9,20.3,20.9,21.2,21.3,21.1,21.1,21.0,21.0,20.2,19.2,18.2,17.8,17.4,17.2,16.9,16.9],
"windspeed_10m":[4.4,4.8,5.0,4.5,4.2,4.7,10.5,13.0,15.3,15.9,15.6,15.8,14.3,12.1,10.7,8.1,7.4,4.1,4.6,1.1,2.3,3.4,4.4,2.9],
"winddirection_10m":[99,103,111,119,121,67,49,46,41,39,29,24,18,17,20,32,23,15,315,288,219,238,215,210]}}
```


Figure 4.40: Open Meteo API JSON Response

The JSON response has two parts, the `current_weather_units` object that con-

tains the current weather data, and the `hourly_units` object that contains the forecasted weather data for the next 24 hours. Each weather data object contais the reqested data types listed below:

- **temperature_2m.** The temperature at 2 meters above ground level, in Celsius degrees.
- **windspeed_10m.** The wind speed at 10 meters above ground level, in meters per second (m/s).
- **winddirection_10m.** The wind direction at 10 meters above ground level, in degrees.

Uncertainty Visualization

Next to each weather data value, the system displays an uncertainty value that indicates the level of uncertainty in the data. The uncertainty value is a percentage that represents the confidence level of the data, and it is incresed for predictions that are further in the future.

Forecasted data slider

The system includes a slider that allows the user to view the forecasted weather data for the next 24 hours. The user can use the slider with a simple grab gesture to select a specific hour, and the system will retrieve the corresponding weather data from the list of weather data objects and display it in the weather panel. The slider is also based on the MRTK3 slider prefab, and it is designed to be easy to use and intuitive for the user. The slider is set to a range of 0 to 23, representing the 24 hours of the forecasted data, and it includes a label that displays the selected hour and date.

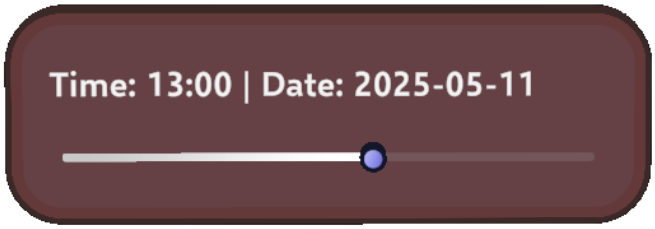

Figure 4.41: Weather forecast slider

4.9.2 Wind Direction Visualization in AR

Along with the weather data panel, the system also includes a wind direction visualization feature that displays the current wind direction in the AR environment. The wind direction is represented by some 3D arrows that are spawned in the AR environment, moving in the direction of the wind.

To implement this feature, the system has WindArrowManager script that has two main functions, the `SpawnArrows` and the `UpdateArrowDirections`.

The `SpawnArrows` function is called when the user toggles the Wind Direction Display button on the weather panel, and it uses a grid spawn logic with given a given grid size and spacing to instantiate the arrow prefabs in the AR environment.

```
// Spawn the grid of arrows centered around the user's position
public void SpawnArrows()
{
    // Get the user's (main camera) position
    Vector3 userPosition = Camera.main.transform.position;

    // Calculate the offset to center the grid around the user
    float gridOffsetX = (gridSizeX - 1) * spacing / 2f;
    float gridOffsetZ = (gridSizeZ - 1) * spacing / 2f;

    // Spawn arrows with the center aligned to the user's position
    for (int x = 0; x < gridSizeX; x++)
    {
        for (int z = 0; z < gridSizeZ; z++)
        {
            // Adjust the position so the grid is centered around the user
            Vector3 position = new Vector3(x * spacing - gridOffsetX, 0, z * spacing - gridOffsetZ) + new Vector3(userPosition.x, 0, userPosition.z);
            GameObject arrow = Instantiate(arrowPrefab, position, Quaternion.identity, windArrowParent);
            arrow.transform.eulerAngles = new Vector3(90, 180, 0);
            windArrowsList.Add(arrow);
        }
    }
}
```

Figure 4.42: Spawn Wind Arrows Function

Then we have the `UpdateArrowDirections` function that is called every time the weather data is updated, and it updates the rotation of all the wind arrows in the AR environment to match the current wind direction. The function takes as input the wind direction value in degrees, and rotates each arrow prefab to correctly point in the direction of the wind.

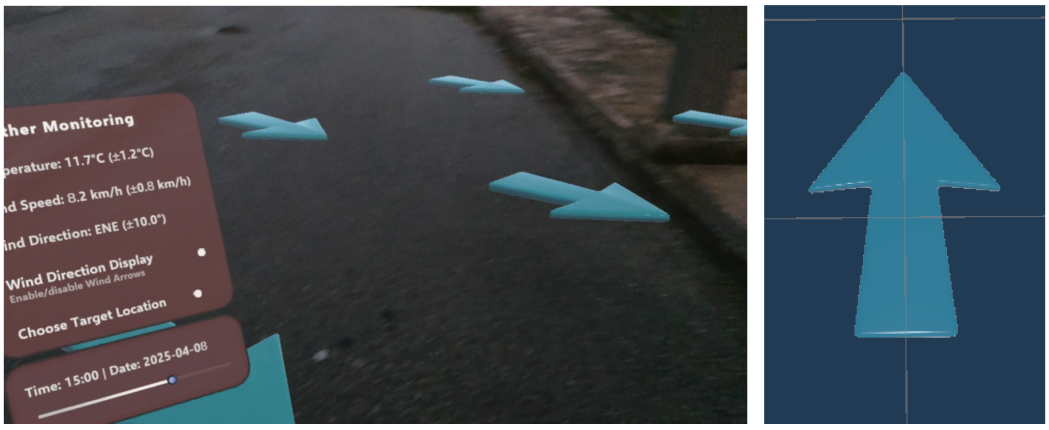

Figure 4.43: 3D Wind Arrows Display

And then for the arrows to conditiuslly move in the direction of the wind, the system uses a simple script attached to each arrow prefab that moves the arrow in the direction of the wind, the arrow moves forward for a fixed distance, and then it moves back to its original position, creating a simple animation effect that simulates the wind direction. The speed of the arrow movement and the distance it moves can be adjusted in the unity inspector, allowing an easy customization of the animation effect.

```
void Update()
{
    // Move the arrow forward in its local forward direction
    transform.Translate(Vector3.left * speed * Time.deltaTime, Space.Self);

    // Calculate the distance traveled from the starting position
    float distanceTraveled = Vector3.Distance(startPosition, transform.position);

    // Check if the arrow has traveled beyond the maximum distance
    if (distanceTraveled >= maxDistance)
    {
        // Reset the arrow's position to the starting point
        transform.position = startPosition;
    }
}
```

Figure 4.44: Wind Arrow Movement Code

4.9.3 Weather Data on Target Location

Another feature that was implemented in the system is the ability to display the current and forecasted weather data on any selected location on the map. This feature was requested on the initial trials. To implement this feature, the system spawns a pointer prefab on the map, which the user can move to the location of interest. The system then Traslates the pointer's position relative to the map, to GPS coordinates using the `WorldToGeoPosition` function provided by the Mapbox SDK, and then calls again the OpenMeteo API using the new GPS coordinates as input parameters. When API response is received, the system parses the JSON response same way that was explained before and displays the requested data in the weather panel.

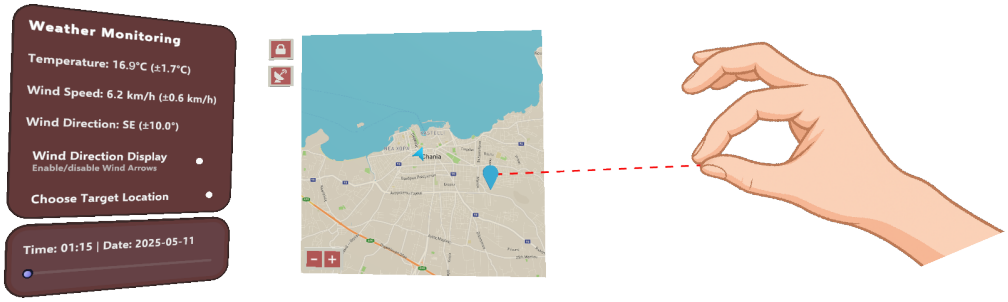

Figure 4.45: Wind Arrow Movement Code

4.10 Conclusion

By utilizing all these features, such the map display with all the information about the team members, POIs, active fire locations and fire spread simulation that are displayed on it and in the AR environment for a more immersive experience, along with the easy to follow navigation system and the weather data monitoring, the system provides users with a comprehensive and effective tool for firefighting operations. Each part of the system is designed to work together seamlessly, and being easy to use and intuitive, allowing users to quickly access the information they need and make informed decisions during firefighting operations. The system is also designed to be extensible, allowing for the addition of new features and improvements in the future, based on user feedback and real-world testing. This chapter has provided an overview of the implementation of the system, including all the main features and functionalities, and the techniques used to implement them.

Chapter 5

Evaluation

5.1 Introduction

This chapter presents the evaluation of the system, including the initial trials and feedback received from users. The evaluation process is crucial to assess the effectiveness and usability of the system in real-world scenarios, and to identify areas for improvement.

5.2 Local Initial Trials

The initial trials were conducted locally in our lab, involing two participants, one firefighting Officer and one urban planning professor. The trials were designed to test all the implemented features of the system, and to gather feedback about the usability and effectiveness of the system in real-world scenarios. The participants were asked to perform a series of tasks using the system and to provide feedback about their experience.

5.2.1 Trials Feedback

The participants provided valuable feedback about the system, focusing on the usability and effectiveness of the implemented features. They mentioned that all the implemented features would be useful for firefighting operations, and they found the system easy to understand and use. They also provided suggestions for improvements and additional features that could enhance the system's usability and effectiveness. Some of the suggestions have already been implemented in the system after the trials, while others are still under consideration for future work.

Implemented Suggestions

- The participants had difficulty using the menu buttons, as some of them were too far from the user, making it difficult to select them with the hand

gestures. So they suggested to place the menu buttons closer to the user for easier access. The system was modified to allow allow the user to directly select the menu buttons with a simple hand gesture, and also enabled the gaze selection feature, which allows the user to select a UI element by looking at it, making it easier to interact with the menu buttons.

- They also requested the ability to display weather information for a selected location on the map, rather than just the user's current location. This feature was implemented in the system after the trials, by lettng the user to move a pointer prefab on the map to the location of interest, to display the weather data for that location.
- The participants expressed a preference for a satellite map style, that would provide a more realistic view of the environment and help them more when using some of the system's features. This feature was also added to the system, by adding a toggle button on the map panel that allows the user to switch between street view and satellite view on the map.

Figure 5.1: Weather information for a selected location Feature

Figure 5.2: Satellite Map Style implementation

Other Suggestions

- They suggested adding more information about the route, souch as if there is a bridge or a tunnel on the route, or the conditions of the path, to help them make more informed decisions about the route to take.
- The participants requested a local wind direction feature, which would provide information about the wind changes effected by the fire in the area.
- They also suggested providing more frequent predictions for the weather data, to help them make more accurate predictions about the fire behavior and progress in the area.

5.3 Summary of the Evaluation

The initial trials provided valuable feedback about the system, and helped to identify areas for improvement and additional features that could enhance the system's usability and effectiveness. The feedback received from the participants was taken into consideration when implementing the system, and many of the suggestions were implemented in the system after the trials. The trials also helped to validate the effectiveness of the implemented features, and to assess the usability of the system in real-world scenarios.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis presented the design and implementation of an Augmented Reality (AR) system for firefighting operations, focusing on providing real-time information about the environment and the team members. The system was designed to be easy to use and intuitive, allowing users to quickly access the information they need and make informed decisions during firefighting operations. The system includes several features, such as a map display with all the information about the team members, Points of Interest (POIs), active fire locations, and fire spread simulation, along with a navigation system and weather data monitoring, along with an AR visualization of the data to provide users with a comprehensive and effective tool for firefighting operations.

The system was evaluated through initial trials with real users, who provided valuable feedback about the usability and effectiveness of the system. The feedback received from the participants helped to validate the effectiveness of the implemented features, and to assess the usability of the system in real-world scenarios, and was taken into consideration to work on some final improvements and additional features that could enhance the system's usability for the users.

However some of the requests and suggestions from the participants are still under consideration for future work, as they require more advanced techniques and technologies to be implemented, such as the local wind direction feature, which would require the use of advanced sensors and algorithms to accurately measure the wind direction in real-time. Also there are some hardware limitations that need to be addressed, such as the size and weight of the HoloLens2 device, which can be uncomfortable for users during long periods of use, along with the fact that the device is not designed to be used in extreme conditions such as high temperatures. These limitations need to be addressed in future work, to ensure that the system can be used effectively in real-world scenarios.

6.2 Future Work

The future work for this project includes several areas of improvement and additional features that could enhance the system's usability and effectiveness. Some of the areas for future work include:

- **Thermal Imaging.** The system could be enhanced by integrating thermal imaging technology, which would help the user see through smoke and identify hot spots in the environment. This would provide users with additional information that could help them make more informed decisions during firefighting operations.
- **Local Wind Direction.** The local wind direction feature could be implemented by using advanced sensors and algorithms to accurately measure the wind direction in real-time, providing users with more accurate information about the wind changes in the area to help them make mor eaccurate predictions about the fire behavior and progress.
- **Integrate with Drones.** The system could be integrated with drones to provide users with real-time aerial views of the environment, helping them to identify hot spots and assess the fire's behavior from different angles. This would provide users with additional information that could help them make more informed decisions during firefighting operations.
- **Integrate with a health monitoring system.** The system could be integrated with a health monitoring system to provide the user with real-time information about the health status of the team members, such as heart rate, body temperature, and other vital signs.
- **Attach a GPS module to the device.** The system could be enhanced by attaching a GPS module to the HoloLens2 device, so the system does not require a smartphone to provide the GPS coordinates of the user. This would allow the system to work independently of the smartphone, providing users with more flexibility and convenience during firefighting operations.

With the implementation of some of these features, the system can become a more comprehensive and effective tool for firefighting and can be used in a wider range of real-world scenarios.

Bibliography

[1] Bhattarai, M., Jensen-Curtis, A. R., & Martínez-Ramón, M. An Embedded Deep Learning System for Augmented Reality in Firefighting Applications. *19th IEEE International Conference on Machine Learning and Applications (ICMLA).*

[2] Steingart, D., Wilson, J., Redfern, A., Wright, P., Romero, R., & Lim, L. Augmented Cognition for Fire Emergency Response: An Iterative User Study. *1st International Conference on Augmented Cognition, 2005.*

[3] Wani, A. R., Shabir, S., & Naaz, R. Augmented Reality for Fire & Emergency Services.

[4] Kang, H.-S., Lee, J.-W., & Choi, S.-M. Combining Augmented and Virtual Reality Experiences for Immersive Fire Drills. *SIGGRAPH Asia 2022 Posters.*

[5] Yang, U. Compact Near-Eye Display for Firefighter's Self-Contained Breathing Apparatus. *ETRI Journal, 2023.*

[6] Dianxi, Z., Danhong, C., & Zhen, G. Design of Intelligent Firefighter Helmet Based on AR Technology. *IOP Conference Series: Earth and Environmental Science.*

[7] Chalimas, T., & Mania, K. Cross-Device Augmented Reality for Fire and Rescue Operations based on Thermal Imaging and Live Tracking. *2023, IEEE International Symposium on Mixed and Augmented Reality Workshops (ISMARW).*

[8] Bailie, T., Martin, J., Aman, Z., Brill, R., & Herman, A. Implementing User-Centered Methods and Virtual Reality to Rapidly Prototype Augmented Reality Tools for Firefighters. *10th International Conference on Augmented Cognition, Toronto, Canada, 2016.*

[9] Lattimer, B. Y., & Hodges, J. L. Intelligent Firefighting. *Handbook of Cognitive and Autonomous Systems for Fire Resilient Infrastructures.*

[10] Li, P., Xia, Q., Wen, X., & Lu, C. X. See through Smoke: Real-time Scene Perception and Navigation on Smart Firefighter Helmet.

[11] Abdelrahman, Y., Henze, N., Knierim, P., Schmidt, A., & Wozniak, P. W. See through the Fire: Evaluating the Augmentation of Visual Perception of Firefighters Using Depth and Thermal Cameras. *2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings.*

[12] García, A. F., Biain, X. O., Lingos, K., Konstantoudakis, K., Hernández, A. B., Iragorri, I. A., & Zarpalas, D. Smart Helmet: Combining Sensors, AI, Augmented Reality, and Personal Protection to Enhance First Responders' Situational Awareness.

[13] Zhu, Y., & Li, N. Virtual and Augmented Reality Technologies for Emergency Management in the Built Environments: A State-of-the-Art Review. *Journal of safety science and resilience, 2021.*

[14] Battistoni, P., Sebillo, M., Di Gregorio, M., Tortora, G., Giordano, D., & Vitiello, G. Wearable Interfaces and Advanced Sensors to Enhance Firefighters Safety in Forest Fires. *Proceedings of the International Conference on Advanced Visual Interfaces, 2020.*

[15] Haniff, D. J., & Baber, C. Wearable Computers for the Fire Service and Police Force: Technological and Human Factors. *Third International Symposium on Wearable Computers, 1999.*

[16] Safranoglou, I. Augmented Reality for Real-time Decision-Making in Flood Emergencies. *2024 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct).*

[17] Oregui, X. Augmented Reality Interface for Adverse-Visibility Conditions Validated by First Responders in Rescue Training Scenarios. *Electronics, 2024.*

[18] Campos, A. Mobile augmented reality techniques for emergency response. *16th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services. 2019.*

[19] Zhang, Kexin, et al. Exploring the design space of optical see-through AR head-mounted displays to support first responders in the field. *2024 CHI Conference on Human Factors in Computing Systems.*

[20] Kuo, Chyi-Gang, et al. Research on the Wearable Augmented Reality Seeking System for Rescue-Guidance in Buildings. *Engineering Proceedings 55.1 (2023).*

[21] Papakostas, Christos, et al. "Measuring user experience, usability and interactivity of a personalized mobile augmented reality training system." *Sensors 21.11 (2021): 3888.*

[22] Koutitas, George, Scott Smith, and Grayson Lawrence. "Performance evaluation of AR/VR training technologies for EMS first responders." *Virtual Reality 25.1 (2021).*