MIXED-REALITY SQUAD-COORDINATION PLATFORM

by

Elias Mahfoud

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Computer Science

Charlotte

2016

Approved by:

_____

Dr. Aidong Lu

_____

Dr. Zachary Wartell

_____

Dr. Shaoting Zhang

ABSTRACT

ELIAS MAHFOUD. Mixed-reality squad-coordination platform. (Under the direction of DR. AIDONG LU)

Coordination and situation awareness are amongst the most important aspects of collaborative emergency response for firefighters, police, military soldiers, and so forth. Generally, command centers are in charge of managing a team, typically using voice communication, cameras, and possibly hand-held devices; and it is ideal to provide effective communication channels among team members too. Recent advancements in wearable display technology have introduced new tools to assist in such tasks.

This thesis builds a mixed-reality platform to assist the coordination of squads when performing tasks in real time. The platform provides tools to deliver visualization and heads up display (HUD) information to squad members, as well as reporting tools that allow squad members to coordinate with each other and their command center. The platform uses mixed reality devices (Microsoft HoloLens® devices in this case) to synchronize member locations and other vital information across the team. Such a system allows dispatchers to efficiently exchange useful visual information, such as targets and paths, with field units. The example results demonstrate that this type of direct visual communication has many advantages over simply relying on voice communication and hand-held devices.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Aidong Lu, and my committee members Dr. Zachary Wartell and Dr. Shaoting Zhang, for giving me such a great opportunity to work on this project. I would also like to thank my supervisor Fred Brillante, and my colleagues Benjamin Reese, Christopher Gonzales, Jonathan Coty, Michael Peddycord, and James Gross, for constantly offering support and constructive advice.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Wearable technology has recently achieved tremendous advancements in many aspects, including processing power, display techniques, smaller and more efficient sensors, and more effective software algorithms. Such advancements enabled the use of augmented and mixed reality in a myriad of fields: gaming, education [9], architecture, facilities management, emergency response and military operations, and many others [8]. Specifically, there has been a recent trend in applying mixed reality in command center environments. Most of the work has focused on utilizing the technology within the command center to reduce the cost of infrastructure, as seen in *Future Mission Systems* ([13] and [12]) developed by BAE Systems [14]. This thesis presents a system which focuses on the use of mixed reality by operatives in the field. The goal is to enhance team coordination in scenarios such as police, military, and firefighting missions.

1.1 Technology Background

Extensive research has been conducted on the history of augmented reality in several survey papers. Azuma [2] wrote a detailed discussion of the state of the AR technology in 1997. In 2015, Billinghurst et al. delivered another extensively detailed coverage of the technology and its available tools at different levels [3]. Also in 2015, Thangarajah et al. reported current methodologies for vision-based tracking in AR [17]. In summary, Mixed reality has been attempted since 1968 by Ivan Sutherland [11]. The concept has

been attempted frequently since then, but the technology was not advanced enough for practical applications. In 2012, Oculus Rift was invented, and with it came a drastic shift in the industry. More and more companies started focusing on virtual reality and augmented reality, including Google, Microsoft, HTC, and Sony. In 2015, Microsoft announced their Windows Holographic platform, demonstrated through their new device HoloLens. Since then, many other competitors have been emerging, such as *Meta 2* and *Magic Leap*.

1.2 Related Work

In 1997, Feiner et al. [5] presented a system which used augmented reality to provide information about the university campus. Information was overlaid through a head-mounted see-through display and a hand-held device. Input was collected from the hand-held device. The system used GPS to determine position, accelerometer to determine head pitch and roll, and a magnetometer to determine head yaw. Tache et al. introduced a similar system [16] in 2012 as a training tool. Their system reports operative locations in the field to the command center, and displays that information on a calibrated CAD model. The command center dispatcher also has the ability to send information such as routes and 3D models to the operatives, which will be overlaid in their field of vision. Another similar system was designed by Huang et al. [6] in 2016. This system, called ARGIS, introduces a precise registration method for displaying GIS data in augmented reality. The precise registration method requires user input through a hand-held device to guide the calibration process and compensate for measurement errors in position and orientation. All of the aforementioned systems are designed to work outdoors and hence the reliance on GPS data.

For indoor systems, Irizarry et al. presented InfoSPOT [7], an AR system designed to work indoors with available devices such as an iPad. The system displays BIM (Building Information Model) data to show infrastructure and other information overlaid in the camera view of an iPad. Predefined markers were placed into the environment and registered in the BIM database, along with their position and orientation. The user scans a marker image and that allows them to identify which area they are located in and view that environment through the device.

Most of these projects relied only on reading data from a centralized database. Some of them allowed for interactive data authoring from the server side, but operatives in the field were still unable to communicate back, other than reporting their location.

On the other hand, many researchers have addressed collaboration between users in immersive, controlled and small environments; which is very suitable for virtual reality. Cordeil et al. proposed a system for collaborative analysis of network connectivity [4], which compared the performance between a CAVE2 and a HMD. Their application used the Unity engine to present and interact with the data, and their findings showed a considerable performance improvement of HMDs, especially taking into account their affordability. An augmented-reality collaboration environment was presented by [15], where a stereoscopic display was projected onto a see-through display, allowing multiple users to simultaneously view the same spatially-aligned model. And recently, a collaboration service based on Microsoft HoloLens was published by Object Theory [1]. It allowed multiple people to view and interact with models while seeing virtual avatars of each other, making it easy to point at objects and communicate remotely.

This thesis combines techniques from the related work to provide a server-client ar-

chitecture supporting two-way communication with the clients, allowing more effective communication with operatives in the field. The system is designed to work indoors. Dispatchers are able to interactively add instructions and models to operatives' views (similar to [16]), while allowing operatives wearing HoloLens devices to interact with the environment and send information back to the server (similar to [4] and [1]).

1.3 Microsoft HoloLens

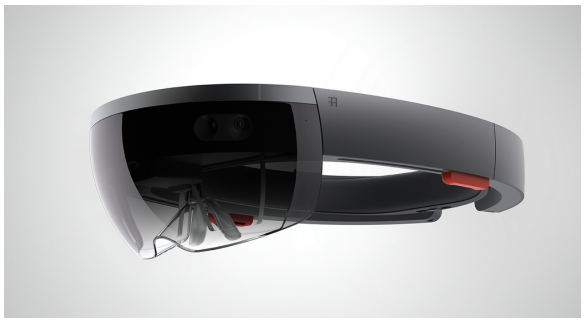


Figure 1: A side view of the Microsoft HoloLens (image courtesy of Microsoft)

Microsoft HoloLens (fig. 1) is the device chosen to implement the squad coordination platform described in this thesis. HoloLens is one of the first untethered mixed-reality devices. It has decent processing power (roughly equivalent to a Surface Pro 2 tablet), and a multitude of sensors and cameras to aid in spatial mapping. HoloLens also offers powerful integration with the popular 3D engine Unity, which greatly facilitates creating interactive 3D holographic applications. The conventional ways of interaction use voice commands, two standard hand gestures, and gaze direction. This remainder of this section briefly describes the concepts used for creating holographic applications with HoloLens.

1.3.1 Coordinate Systems

HoloLens supports three frames of reference to aid in placing holograms in the world: a stationary frame of reference (the default), an attached frame of reference, and a head-locked frame of reference. The stationary frame of reference is designed to keep objects around HoloLens stationary as the device moves. It attempts to stay stable in the vicinity of the device, while further coordinates may suffer from drift and inaccuracies. The stationary frame of reference is typically created at application start, and maintained during the application lifetime.

The attached frame of reference follows the user's position but stays at a constant orientation. The HoloLens documentation advises that this coordinate system should be used to display fallback UI when the device loses tracking.

The head-locked frame of reference is rigidly attached to the user's head, following position and rotation (such as a static HUD). The HoloLens documentation advises against using this frame of reference because it is uncomfortable to users.

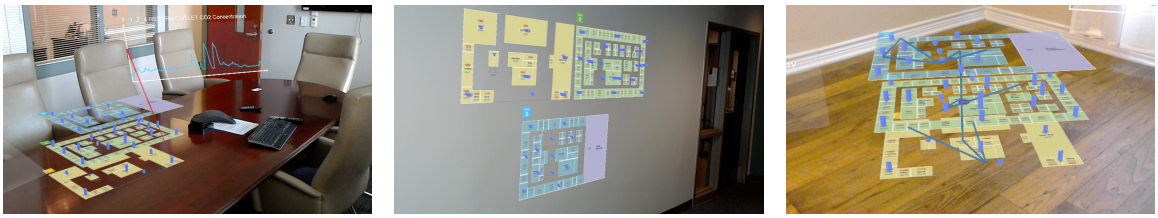
1.3.2 Spatial Anchors

To address drift issues with the stationary frame of reference, HoloLens offers a mechanism called a *spatial anchor*. Spatial anchors contain information about the environment that allows the device to accurately locate the point relative to the real world. Objects placed under a spatial anchor remain in their precise location in the real world, at the cost of possible drift from the stationary frame of reference. Spatial anchors can also be serialized and shared between devices, allowing them to share points between their different coordinate systems and to show objects at the same real-world location.

CHAPTER 2: PRELIMINARY RESULTS

Several HoloLens visualization and visual analytics projects have been created leading up to the final system. These projects served as an excellent platform to learn about the device, its features, and its limitations. Many of the tools and techniques created for these projects were integrated in the final system. This chapter describes three projects that have been implemented using HoloLens and the Unity engine.

2.1 Floor Plan Visualization



(a) Room time line **(b)** Larger floor plans **(c)** User paths

Figure 2: Floor plan data visualization

This project explores visualizing floor plans in a mixed-reality context. It overlays visualization, allowing an investigator to effectively assess security data and building telemetry. The project displays floor plans as a set of layers, placing bars at various locations to represent sensor readings (fig.2a). Three-dimensional paths can also be visualized to show tracked user locations across floors (fig. 2c). Each floor can also be copied, enlarged, and placed on walls for further inspection (fig.2b). A user can use voice commands to display a detailed time line for any location. This work was the basis for the world-in-miniature

feature implemented in the final system.

2.2 Using Gaze Input to Explore Frames of a Time Series

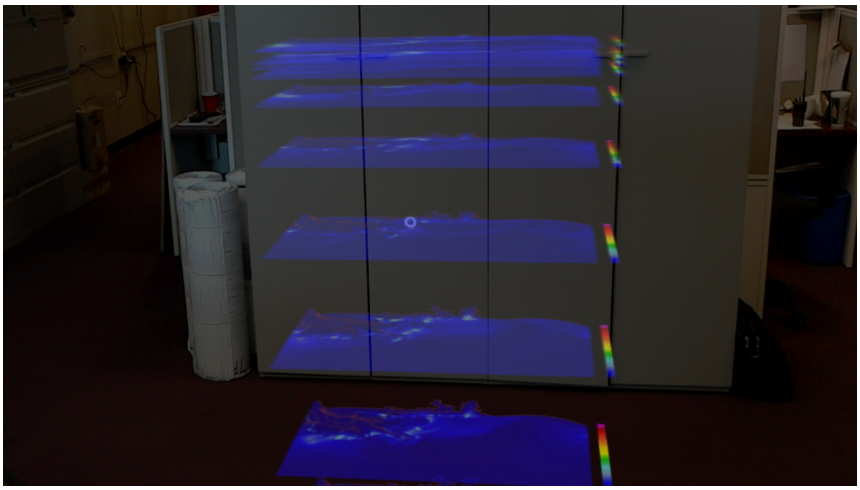
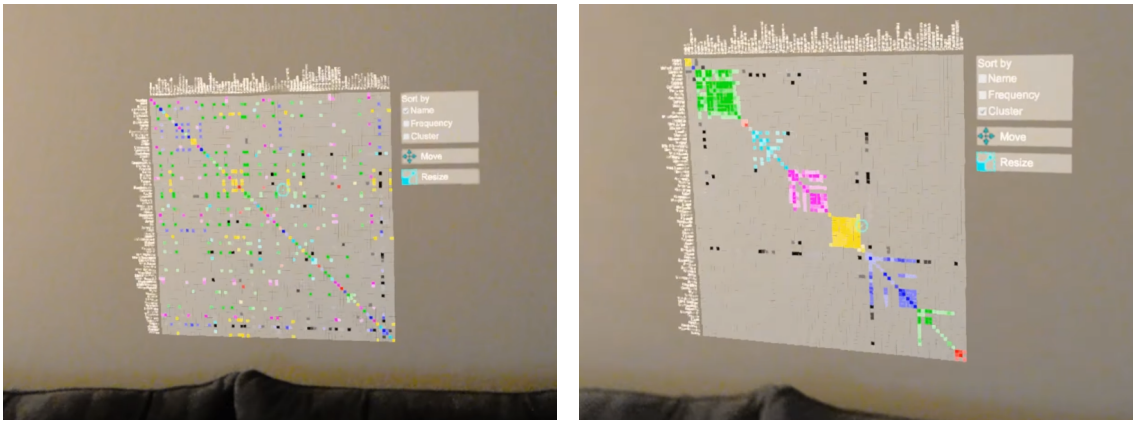


Figure 3: Utilizing gaze input to quickly explore a stack of images.

This project experiments with a new way to inspect a series of images, by displaying them as a vertical stack (fig. 3) and dynamically separating them further at the point where the user is looking. This facilitates quickly scanning through the time line by vertically moving the gaze point, and quickly picking a frame for further inspection by pausing the gaze at it for a short period of time.

2.3 Adjacency Matrix Visualization

This project visualizes a graph using an adjacency matrix (shown in fig. 4). Given the sheer number of nodes required by this type of visualization, the HoloLens hardware was not able to accommodate the conventional way of creating scene graphs in Unity. Instead, a custom particle system was created. The particle system allowed individual point sprites to be very efficiently animated, achieving interactive frame rates. The resulting program performs at 60 fps with smooth animations when switching axis orders. It also utilizes



(a) **(b)**

Figure 4: Screenshots from the adjacency matrix visualization program. (a) shows axes sorted by name, and (b) shows axes sorted by cluster.

voice commands to rearrange, move, and resize the graph. The technique developed for this project has been used to render targets in the world-in-miniature view without hindering the HoloLens performance.

CHAPTER 3: SYSTEM DESIGN AND IMPLEMENTATION

3.1 Overview

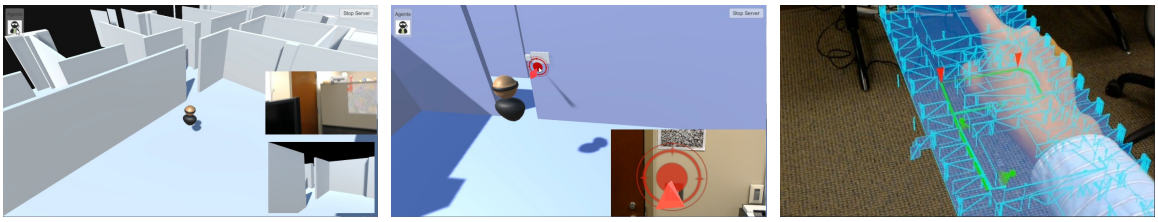


Figure 5: System screenshots (mixed-reality footage is superimposed for comparison).

The system is built using the Unity3D engine. It is designed to work in a command center, utilizing mixed-reality devices to assist in operations such as firefighting, police and military operations, and emergency evacuations. The system has a server component used by the dispatcher at the command center, and a set of HoloLens devices used by operatives in the field. Various information can be transmitted back and forth between the server and the operatives. Each device can synchronizes its location (in its stationary frame of reference) with the server and other devices. Operatives can also issue commands to describe changes in their environment, for example marking a location as a new target, or requesting rally point at a location they see. The dispatcher has a bird's eye view of the situation. They can navigate through the scene which has the 3D CAD model loaded, and operative characters shown at their corresponding locations. The dispatcher is able to create targets and paths for operatives to follow.

3.2 System Features

The system implements the following features:

- Support for a server and multiple connected HoloLens devices
- Operative locations are synchronized in real time between all devices and the server
- Dispatchers can send target locations and paths to operatives in the field
- Operatives can create targets within line of sight using voice commands
- Operatives have a world-in-miniature view (WIM) to locate teammates, view all targets and paths, and create new targets that are not within line of sight.

The implementation of these features will be discussed in the remainder of this chapter.

3.3 Support for Multiple Connected Devices

The system uses UNET, Unity's high-level networking API (HLAPI). This API is server-centric. It requires server-client communication and does not support peer-to-peer communication. The HLAPI determines which objects can be controlled by which clients using authority (see appendix A for details).

While the HLAPI offers many automatic synchronization features, it assumes that all clients and the server share the same coordinate system; however, due to the nature of HoloLens's inside-out tracking, each HoloLens device has its own independent coordinate system (see fig. 6). The different coordinate systems prevent direct use of UNET's location synchronization.

To resolve this problem, a common coordinate system must be used between the server and all connected clients. Several approaches can be followed to establish a common coordinate system. The official HoloLens tutorials use a spatial anchor created by the first

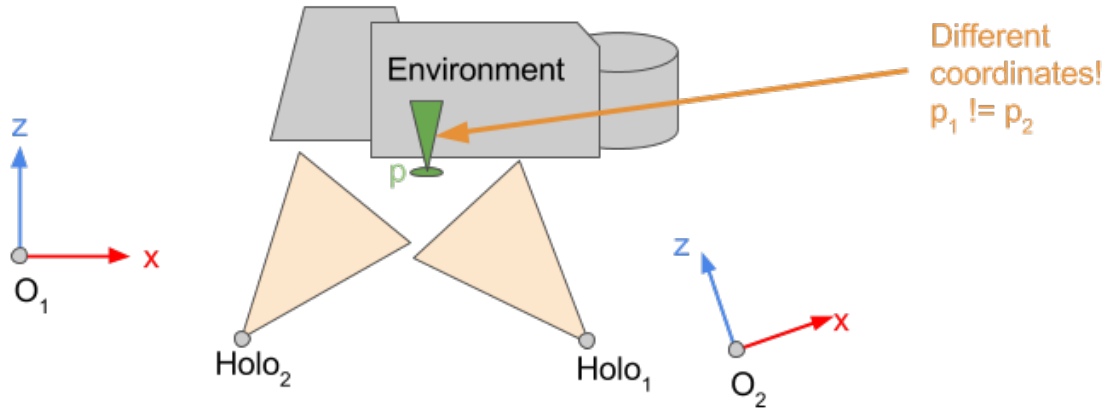


Figure 6: The challenge of having different coordinate systems.

connected client as the common coordinate system. However, with this approach, it is not possible to associate the spatial anchor with a 3D CAD model. Another approach is to use an external, predefined location that has a corresponding point in the CAD model as the common coordinate system. The solution implemented in this thesis uses Vuforia to locate this external predefined point. The CAD model includes an object with the tag 'Origin' at the corresponding point (see fig. 7).

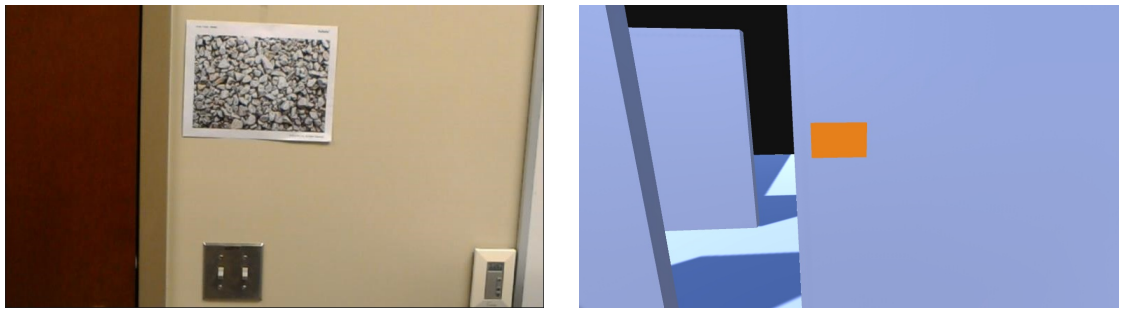


Figure 7: The 'Stones' Vuforia marker (left) and the Origin object at the corresponding point (right).

3.3.1 Using the Vuforia Library

Vuforia is an augmented reality library which uses computer vision to locate predefined images in 3D space. It offers strong integration with the Unity engine, and its latest version

supports HoloLens devices. When a new HoloLens device starts the application, the offline Unity scene is loaded. The scene has an active Vuforia camera configured to locate a predefined pattern (Vuforia target). When that target is located within the camera view, the event handler computes the camera offset and connects to the server, while also loading the online Unity scene.

Several factors impact coordinates when loading a new scene, affecting the way the camera offset is computed in the new coordinate system. First, the HoloLens world position is reset to $(0, 0, 0)$, while the rotation is only reset around the Y axis since HoloLens can compute the correct X and Z rotations from the accelerometer. This means that to compute the initial camera coordinates in the common coordinate space, the position and rotation of the camera have to be computed differently (see fig. 8). Let T_{cam} be the transform of the HoloLens camera in its local coordinate system when the image target is detected, and let T_{img} be the image target's transform in the local coordinate system. The position offset of the camera is computed as follows:

$$T_{offset} = T_{img}^{-1} T_{cam}$$

The translation component of T_{offset} is the initial camera position in the common coordinate system. To compute the initial camera rotation in the common coordinate system, only the rotation around the y axis is needed. Let Z'_{cam}, Z'_{img} be the projections of the Z vectors of T_{cam} and T_{img} onto a horizontal plane, respectively. The rotation around the Y axis can be easily computed by simply computing a quaternion q that rotates from Z'_{img} to

Z'_{cam} as follows:

$$q_{xyz} = Z'_{img} \times Z'cam$$

$$q_w = Z'_{img}.Z'_{cam} + \sqrt{len(Z'_{img})^2 len(Z'_{cam})^2}$$

Notice that the square length of the vectors is used since it is faster to compute, then the square root of their multiplication is taken.

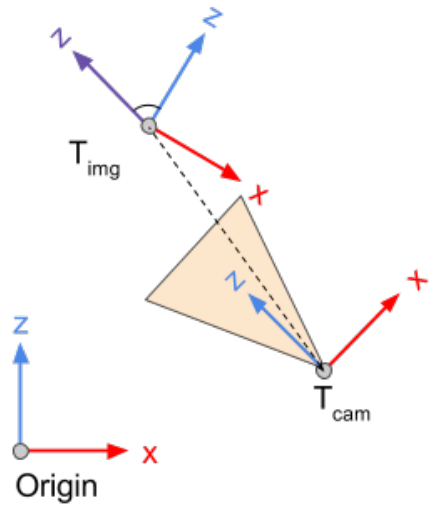
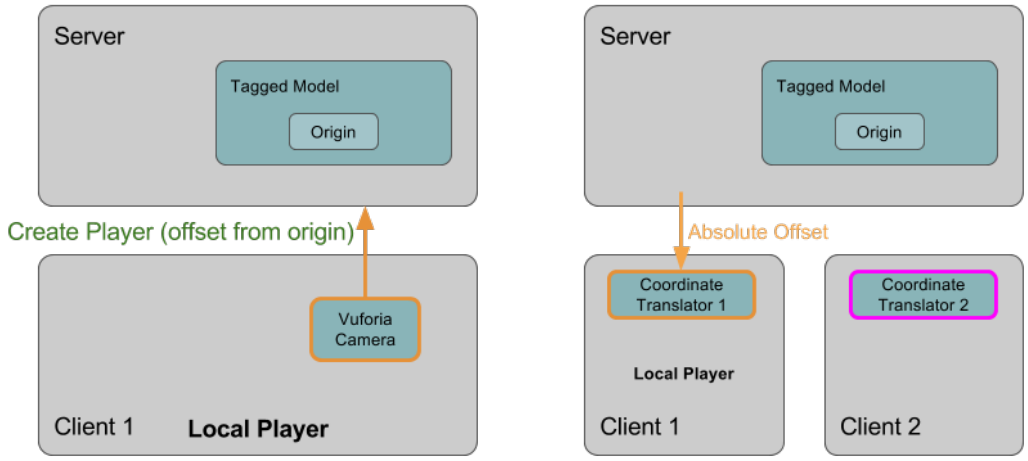


Figure 8: Computing the initial camera offset in the common coordinate system. The position is the offset between the camera transform, T_{cam}, and the image target transform, T_{img}. The rotation is the angle between the projections of the Z vectors of these transforms on a horizontal plane.

The computed transform is relative to the image target's transform in the CAD model. To compute the final initial offset for a client, the server multiplies the relative offset by the corresponding image target's transform in the CAD model. The resulting final offset is sent back to the client to be used to translate coordinates between its local coordinate system and the common coordinate system. The workflow is demonstrated in fig. 9.



(a) Client sends image target offset **(b)** Server sends final offset

Figure 9: The workflow for computing the initial client transform in the common coordinate system. **a)** shows client1 before it connects to the server. The offline scene is loaded and the Vuforia camera is active. When the image target is located, a request is sent to the server to connect the client, and use the calculated offset. **b)** shows the server replying to client1 with its absolute offset, which will be used to translate coordinates for that client. Notice that client2, which is already connected, has a different translator (translator2), since client2 has a different initial offset.

3.4 Synchronizing Operative Locations

After a client connects to the server and receives its absolute offset in the common coordinate system, synchronizing pose is relatively straightforward using UNET. It is not possible to directly use UNET's NetworkTransform to automatically synchronizes the pose of each player; however, it is possible to implement a custom network transform synchronization logic which utilizes lower level constructs such as syncvars (see appendix A for details). The custom network transform uses syncvars to synchronize local position and local rotation independently. It only sends changes to the server if they exceed a configurable threshold. Whenever the threshold is exceeded, the custom network transform converts the coordinates into server space and sends them using a command. The server then automatically synchronizes the changed syncvars across all clients. With a syncvar hook, clients

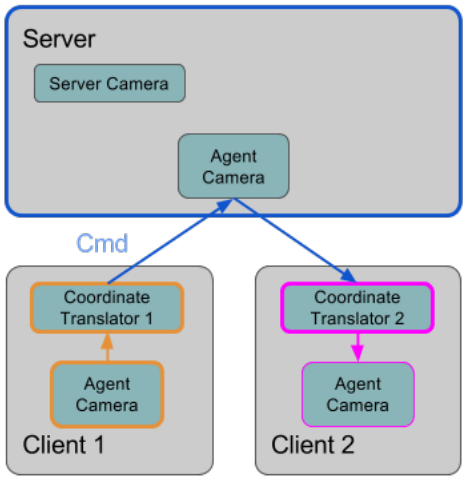


Figure 10: Synchronizing pose across the network. Client 1 translates its coordinates (represented in yellow) into server coordinates (represented in blue) and sends them through a command to the server. The server synchronizes these values to other clients. Client 2 receives the server coordinates, and translates them in a synchvar hook to its local coordinates (represented in magenta).

are able to convert received coordinates into their local space to show objects at the correct locations. A diagram of the process is shown in fig. 10.

It is also worth noting that the custom network transform also interpolates the position and rotation between received coordinates to achieve smooth animation. This custom network transform is used to synchronize the location of all individual objects (all characters representing operatives, and all targets).

3.4.1 Operative Character Animation

Each operative is represented with a character that mimics the head movement of that operative. The synchronized position and rotation are applied to the character head after interpolation, achieving smooth animation of the head. However, to make the characters look closer to a person instead of floating heads, a body section is needed. The body needs to maintain an upright position while the head is moving to make it appear more natural.

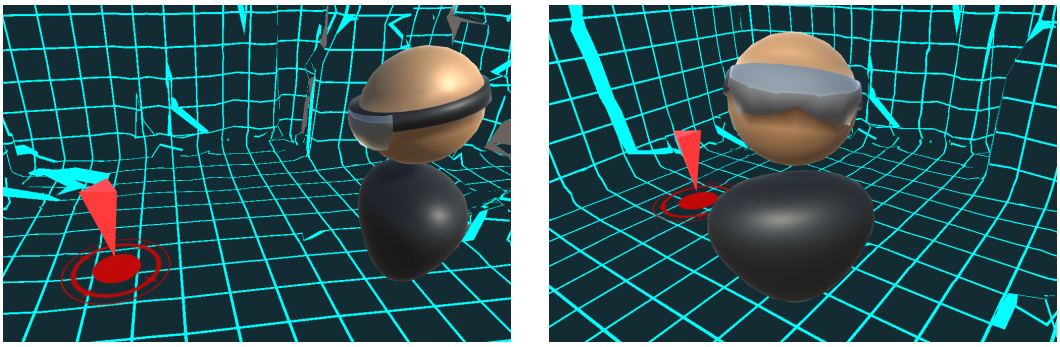


Figure 11: The character representing operative locations in the field.

This is accomplished with a unity behavior attached to the body. The script has, as a parameter, an offset to determine how far below the head the body should be (to simulate a neck). The script moves the body to that location below the head, and it only rotates it vertically to face the same direction the head is looking. The result simulates character movement more naturally and is much more expressive than what only a floating head can be (see fig. 11).

3.5 Creating Targets

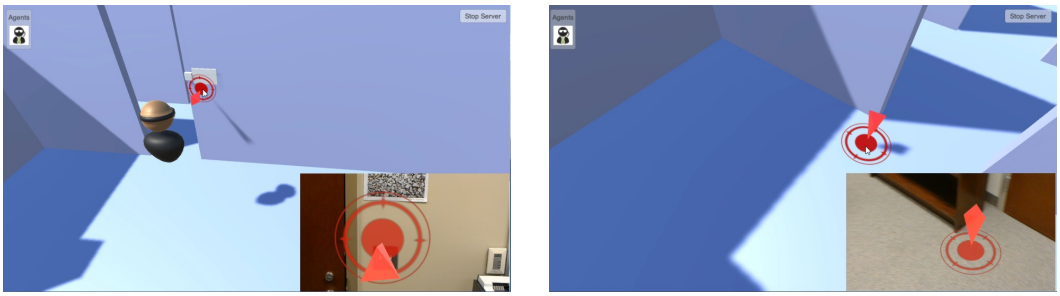


Figure 12: Targets shown on the server and in mixed-reality.

Targets can be created by dispatchers on the server side, or by operatives in the field (see fig. 12). In both cases, the server is responsible for creating and controlling the target instances. The dispatcher can left-click anywhere on the 3D CAD model to create a target. The target's location is synchronized across clients using the same custom network

transform used for the characters.

For operatives to create targets, they issue a voice command (“Create Target”), which sends a command to create the target. The server raycasts the operatives head direction and creates a target at the hit point with the model (see fig. 13).

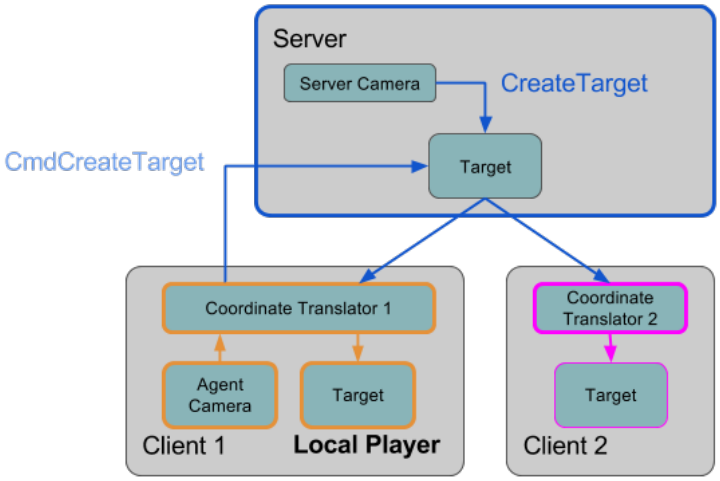


Figure 13: Workflow for creating targets. Targets can be created directly on the server from the server camera object when the dispatcher left-clicks on the model, or they can be created by operatives via a server command when they issue the voice command “Create Target”.

3.6 Creating Paths

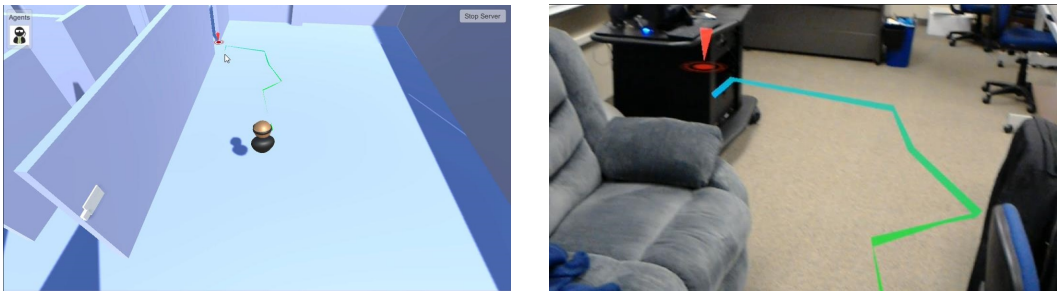


Figure 14: Paths shown on the server and in mixed-reality.

Dispatchers can create paths on the server. Right-clicking on the 3D CAD model initializes a path instance on the server (which is not synchronized yet). Subsequent left clicks

add points to that path, until another right click is issued, which terminates the path and synchronizes it across the network clients (see fig. 14).

3.7 World in Miniature

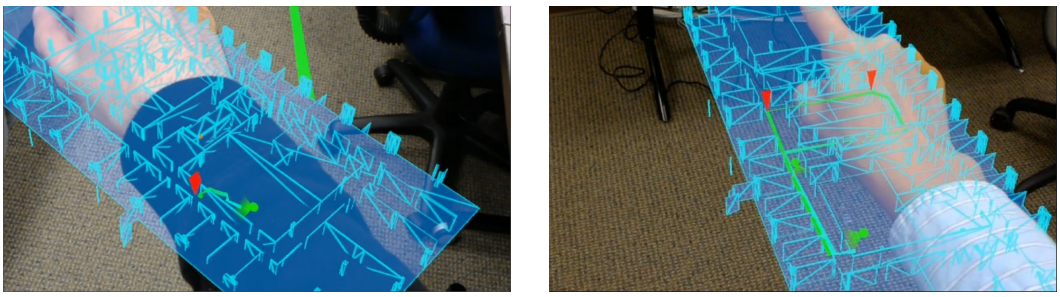


Figure 15: World-in-miniature view.

The system also supports a world-in-miniature (WIM) view to help improve operatives' situational awareness. Stoakley et al. explored the use of a WIM in VR [10], and concluded that WIMs offer many advantages and are intuitively used by users. The WIM implemented in this system shows real-time positions and rotations of operatives in the field, as well as positions of all targets and paths (see fig. 15). It utilizes HoloLens's hand tracking to show it hovering over the operative's hand when it is within view. The WIM can be rotated using the other hand to view it from a better angle. Finally, the WIM allows operatives to create targets at locations that are not within their line of sight, by pointing their gaze point at the desired location and performing a tap gesture.

3.7.1 Using a Particle System to Improve Performance

Several performance issues arise when dealing with the WIM. Simply displaying individual instances to represent all the operatives as well as all the targets pauses a significant performance bottleneck. Since it is important to see operatives' positions and rotations at all times, it is necessary to use individual instances to represent them. Paths are simply

lines so the same technique is used to render them on the WIM as in the full-size world. Targets, on the other hand, can be simplified to positions only. This allows the system to use a custom particle system containing all targets instead of an individual instance for each target. This approach is significantly faster and allows rendering hundreds of targets without a significant frame rate hit. This performance optimization has been adopted from the adjacency matrix project.

3.7.2 Tracking Objects on the WIM

To simplify tracking objects on the WIM, a unity behavior is developed to automate the process. It holds three parameters: a reference to the model use to represent the object on the WIM, a boolean to indicate whether a separate instance should be created or a particle system point, and a color for the particle system point. When the server instantiates the object on a client, this custom behavior adds the object to the WIMManager for tracking along with the appropriate parameters. Then in the LateUpdate function, the WIMRenderer iterates over tracked objects and uses the specified parameters to update the WIM. LateUpdate is used to guarantee that all objects have moved to their location in the frame before the WIM is updated, which yields more accurate positions. The general structure of the WIM system is shown in fig. 16.

3.7.3 Hand Tracking

The WIM uses HoloLens's hand tracking functionality to show and manipulate the view. The HandTracker class registers for the tracking events fired by the HoloLens API. These events are SourceDetected (for when a new interaction source or hand is detected), SourceUpdated (for when a hand is moved), and SourceLost (for when a hand is not visible any-

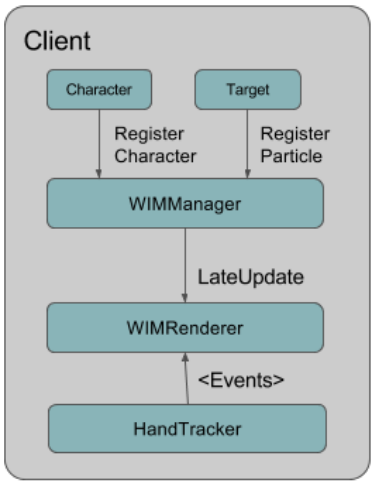


Figure 16: General structure of the WIM system.

more). Each of these events reports the corresponding hand ID and world position (rotation is not tracked by HoloLens). Using these events, the hand tracker can determine how to manipulate the WIM object.

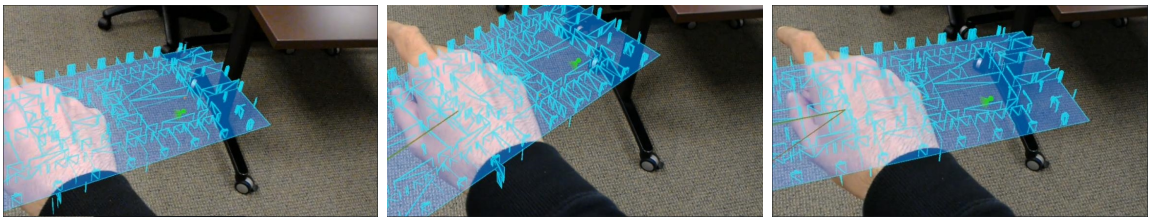


Figure 17: WIM interactions, from left to right: display WIM when first hand is shown, show initial angle when second hand is shown (off frame), and show current rotation while second hand is moving (off frame)

The hand tracker supports the use of two hands to manipulate the WIM. When the first hand is shown, the tracker saves its ID, activates the WIM, and moves it to the tracked position. Subsequent updates of the hand with that ID are used to update the WIM position. When a second hand appears (while the first hand is still tracked), the tracker saves its ID and its initial position. Subsequent updates of the hand with the second ID are used to compute the angle of rotation to rotate the WIM (see fig. 17).

It is worth noting that HoloLens only starts tracking hands when it recognizes the beginning of a standard gesture, which is either an extended index finger or the beginning of a bloom gesture. An open palm or any other gesture does not trigger tracking events.

CHAPTER 4: CONCLUSION AND FUTURE WORK

4.1 Conclusion

This thesis presented a team coordination system which utilizes networked HoloLens devices connected to a server in a command center. The system improves team effectiveness in the field by allowing visual communication in mixed-reality between operatives and the command center. Situational awareness is greatly enhanced with real-time synchronization of operative locations and the use of a WIM. Targets can be created and synchronized by dispatchers and operatives in the field, and paths can be created by dispatchers to easily guide operatives. The thesis also explored the strengths and limitations of current mixed-reality technology. The system is expected to be improved to adapt newer mixed-reality technology and eventually serve as a robust platform to build extensive coordination applications in ubiquitous environments.

4.2 Future Work

Many features can be added to the system to further its effectiveness:

- The use of high-quality CAD models allows server-side automatic path computation, allowing dispatchers to quickly create paths by specifying start and end points, as well as operatives to request a path to a target using the WIM.
- A video feed would be useful in some situations, but this requires improvements in both HoloLens and the Vuforia library (the current version of Vuforia does not

support mixed-reality capture with spatial mapping).

- Coordinate-system drift occurring from large movements should be addressed by realigning whenever a known anchor point is encountered.
- HoloLens has the ability to scan the environment an create a 3D model of it. This ability can be useful when encountering an area not included in a CAD model. While this feature has already been implemented in this system (as seen in fig. 18), it has been intentionally disabled since transmitting that much data to the server is very resource-intensive. Further work should focus on optimizing and polishing this feature.

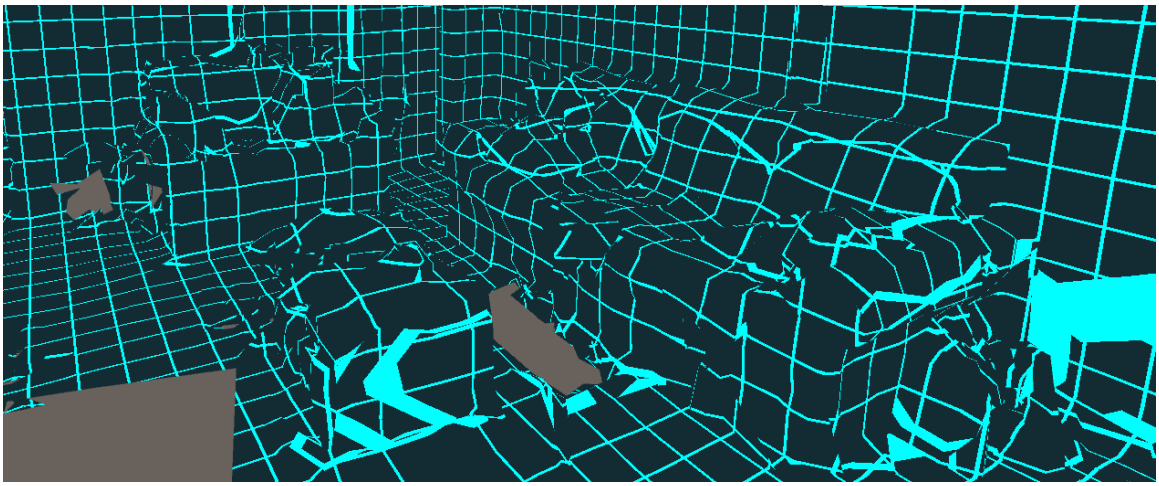


Figure 18: The environment captured by HoloLens.

We plan to continue the work by investigating the above directions.

APPENDIX A: UNITY NETWORKING CONCEPTS

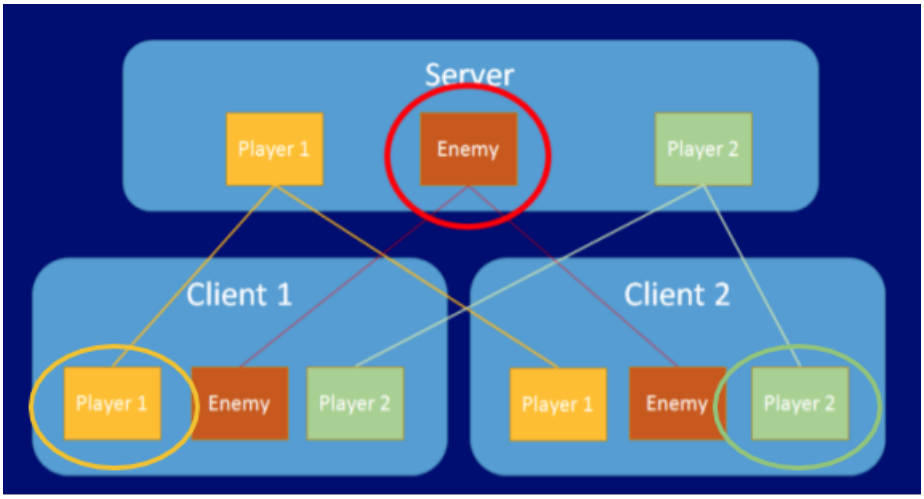


Figure 19: UNET's object authority in network transforms (figure from the Unity Documentation Site). Player 1 is associated with client 1 making it a local player for that client. As a result, player 1's movement is controlled on client 1 and then synchronized with the server and client 2. The same is true for player 2 and client 2. Some objects have server authority (such as enemies). These are controlled by the server and then synchronized to all clients.

Unity 5 provides an advanced networking API called UNET. The core concept is the player object. Each connected person has an associated player object. Each player has instances that exist on the server and all clients. The player instance in its associated client is a local player. Only local players have authority to send commands to player-owned objects on the server, and not to other player instances. This concept of authority helps in determining how objects should behave, such as the network transform (see fig. 19).

UNET also offers the syncvars, which are used extensively in this thesis. Syncvars are variables that, when changed on the server, are automatically synchronized with all clients. Players change syncvars by sending commands. Syncvars also allow attaching hooks, which are callbacks invoked on clients when a new value is sent from the server. This mechanism is used in this thesis for coordinate translation and many other features.

REFERENCES

[1] A mixed-reality collaboration service. objecttheory.com/work, 2015. Accessed: 2016-09-25.

[2] R. T. Azuma. A survey of augmented reality. *Presence: Teleoper. Virtual Environ.*, 6(4):355–385, Aug. 1997.

[3] M. Billinghurst, A. Clark, and G. Lee. A survey of augmented reality. *Found. Trends Hum.-Comput. Interact.*, 8(2-3):73–272, Mar. 2015.

[4] M. Cordeil, T. Dwyer, K. Klein, B. Laha, K. Marriot, and B. H. Thomas. Immersive collaborative analysis of network connectivity: Cave-style or head-mounted display? *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–1, 2016.

[5] S. Feiner, B. MacIntyre, T. Hollerer, and A. Webster. A touring machine: Prototyping 3d mobile augmented reality systems for exploring the urban environment. In *Proceedings of the 1st IEEE International Symposium on Wearable Computers*, ISWC '97, pages 74–81, Washington, DC, USA, 1997. IEEE Computer Society.

[6] W. Huang, M. Sun, and S. Li. A 3d gis-based interactive registration mechanism for outdoor augmented reality system. *Expert Systems with Applications*, 55:48 – 58, 2016.

[7] J. Irizarry, M. Gheisari, G. Williams, and B. N. Walker. Infospot: A mobile augmented reality method for accessing building information through a situation awareness approach. *Automation in Construction*, 33:11 – 23, 2013. Augmented Reality in Architecture, Engineering, and Construction.

[8] W. L. Roux. The use of augmented reality in command and control situation awareness. *Scientia Militaria - South African Journal of Military Studies*, 38(1), 2011.

[9] M. E. C. Santos, A. Chen, T. Taketomi, G. Yamamoto, J. Miyazaki, and H. Kato. Augmented reality learning experiences: Survey of prototype design and evaluation. *IEEE Transactions on Learning Technologies*, 7(1):38–56, 2014.

[10] R. Stoakley, M. J. Conway, and R. Pausch. Virtual reality on a wim: Interactive worlds in miniature. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 265–272, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[11] I. E. Sutherland. A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), pages 757–764, New York, NY, USA, 1968. ACM.

[12] B. Systems. Future mission systems - demo. youtube.com/watch?v=4X7hQEEX4sM, 2015. Accessed: 2016-09-24.

[13] B. Systems. Future mission systems - explanation. youtube.com/watch?v=Z7XczmdYsVA, 2015. Accessed: 2016-09-24.

[14] B. Systems. Bae systems. baesystems.com/en/home?r=US, 2016. Accessed: 2016-09-24.

[15] Z. Szalavári, D. Schmalstieg, A. Fuhrmann, and M. Gervautz. "studierstube": An environment for collaboration in augmented reality. *Virtual Reality*, 3(1):37–48, 1998.

[16] R. Tache, H. A. Abeykoon, K. T. Karunanayaka, J. P. Kumarasinghe, G. Roth, O. N. N. Fernando, and A. D. Cheok. Command center: Authoring tool to supervise augmented reality session. In *2012 IEEE Virtual Reality Workshops (VRW)*, pages 99–100. IEEE, 2012.

[17] A. Thangarajah, J. Wu, B. Madon, and A. K. Chowdhury. Vision-based registration for augmented reality-a short survey. In *2015 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, pages 463–468, 2015.