Networked Architectures for Localization-Based Multi-User Augmented Reality

Jiasi Chen, K. K. Ramakrishnan, Aditya Dhakazl, and Xukan Ran

The authors outline several architectural alternatives for localization in multi-user augmented reality and their applicability to a number of important usage scenarios.

Abstract

Multi-user augmented reality (AR) seeks to enhance the user experience in shared activities, whether collaboration for training, entertainment, or safety applications such as autonomous driving and driver assistance. AR involves a number of tasks that require both sensing and complex computations. A key task is localizing an AR device in the real world, in order to render the virtual holograms at the correct locations on the display. With cloud computing moving closer to the edge, a number of architectural options are available to aid in these complex computations, ranging from performing all of the computation at the end device to moving it all to the cloud, each with a different level of dependency on the communication link. In this work, we outline several architectural alternatives for localization in multi-user AR and their applicability to a number of important usage scenarios. The considered usage scenarios (entertainment, autonomous vehicles, and medicine) reflect a range of latency and spatial accuracy requirements. Within this context, we discuss our recent work on an edge-cloud-centric solution — SLAM-Share — and its applicability to various use cases. In addition, we evaluate its resilience to variations in communication delay. Overall, this article seeks to provide an overview of network architectures for localization-based AR to inform the design of future AR systems.

Introduction

Augmented reality (AR) is a driver of the next wave of "killer apps" in mobile computing. It is revolutionizing the way we live by enhancing our perception of the real world with virtual holograms. AR has found applications in training, education, automotive applications, and entertainment (e.g., Pokemon Go, IKEA Place). In many applications, the virtual holograms are anchored to specific locations in the real-world environment, providing the user with a seamless experience integrating the real and virtual worlds. While much of the current focus has been on providing a high-quality user experience for single-user AR, multi-user AR is a natural extension of the AR paradigm, where multiple users participate in a joint AR session and collaborate and interact with the same set of virtual holograms. This provides a much richer shared experience among the users. There may be numerous applications of multi-user AR, including safety applications such as autonomous driving or driver assistance, collaborative virtual workplaces, and remote collaborative surgical training. However, multi-user AR introduces several technical challenges, because of the need for all the users involved in the activity to share the experience concurrently in real time.

Creating these shared experiences requires several different computations whose results must be communicated between devices. Our article focuses on computations related to device localization, a key component in AR. AR devices need to localize themselves (i.e., determine their own position and orientation in the real world) in order to render the holograms on the display at the right location in the user's field of view. With slow or inaccurate localization, holograms can be rendered at the wrong place on the display, or not in time (see Fig. 1 for an example). This article considers visual-inertial localization that is done by taking sensor readings from a device's camera and inertial measurement unit (IMU), which are then processed through Simultaneous Localization and Mapping (SLAM) algorithms. SLAM results from multiple devices must be communicated and merged together to create a shared AR experience, with low latency and high spatial accuracy. Making the experience high quality (fast and accurate) requires synergistic cooperation between the computation and communication components in a multi-user AR system.

With the potential of extensive Edge Cloud (EC) deployments, intended to be much closer to the user to support various latency-sensitive applications, it is worthwhile to examine how an EC can support various computational tasks for multi-user AR and meet application requirements. The computing power at an EC can be a great enabler for AR in general, and particularly for multi-user AR, by dividing the computations between the end-user's device and the more powerful servers residing in the EC. The EC can also be a convergence point for merging and synchronizing the results of the computations. Different divisions of labor between the EC and AR devices are possible, from EC-centric (where most computations are performed on the EC) to purely on-device operation (where most computations are performed on the end devices), as well as somewhere in between [1–3]. Where computations are performed impacts what is communicated and hence the communication latency; for example, the EC-centric architecture performs fast computations but requires low-de-

ISSN: 0163-6804
Digital Object Identifier: 10.1109/MCOM.003.2300275

The authors are with the University of California, Riverside, USA.

lay, high-throughput network connectivity to the EC to enable a responsive multi-user AR system that meets the quality-of-experience that users expect. Different system architectures provide different capabilities depending on the network and device capabilities, and the choice of architecture must be carefully considered.

The goal of this article is to outline possible networked system architectures and their trade-offs in the context of different multi-user AR use cases. Specifically:

- We provide background on the role of localization with cameras and IMUs in single and multi-user AR applications. Two key computation components of localization are tracking and mapping. We illustrate the impact of the latency and accuracy of these tasks on the AR display.
- We discuss the networked system architectures that can support the computations required for localization. We describe three architectures ranging from fully edge-cloud-assisted to pure on-device operation.
- We discuss three use cases for AR — entertainment, autonomous vehicles, and medical training — including their latency and spatial accuracy requirements, and the ability of the networked architectures to meet these requirements.

Background on AR Localization

AR Platform Architecture

AR platforms consist of multiple layers, as shown in Fig. 2. At the bottom, we have various sensors, such as a gyroscope, accelerometer, visible light camera, depth camera, microphone, and eye tracker. In the middle, we have the AR platform, which typically includes a game engine (e.g., Unity, Unreal) that handles the physics of holograms and rendering, similar to computer games. There is also AR-specific functionality, such as device localization, scene understanding, hand tracking, gesture detection, and voice commands. These functions are implemented through a combination of general libraries (e.g., OpenXR, MRTK) and device-specific libraries (e.g., Oculus Integration Package).

The focus of this work is on the localization of AR devices. Localization is the ability of a device to determine its own location and orientation in the real world. The accuracy and speed of the localization can greatly affect user experience. For instance, suppose that a pedestrian is in the intersection at location (x, y, z) in Fig. 1. A blue car arrives at the intersection, detects the pedestrian at location (x, y, z), and sends the coordinates to the purple and red cars, who are about to turn and hit the pedestrian due to poor visibility from the trees. When the red car arrives at the intersection, it is slow to localize and determine where (x, y, z) is with respect to itself and render the hologram on its AR windshield. This may cause the red car's driver to miss seeing the pedestrian hologram and cause a collision. Inaccurate localization by the purple car causes the pedestrian hologram to appear with a 1 meter offset on the heads-up display, also potentially leading to a collision. Thus, fast and accurate localization is crucial for effective AR.

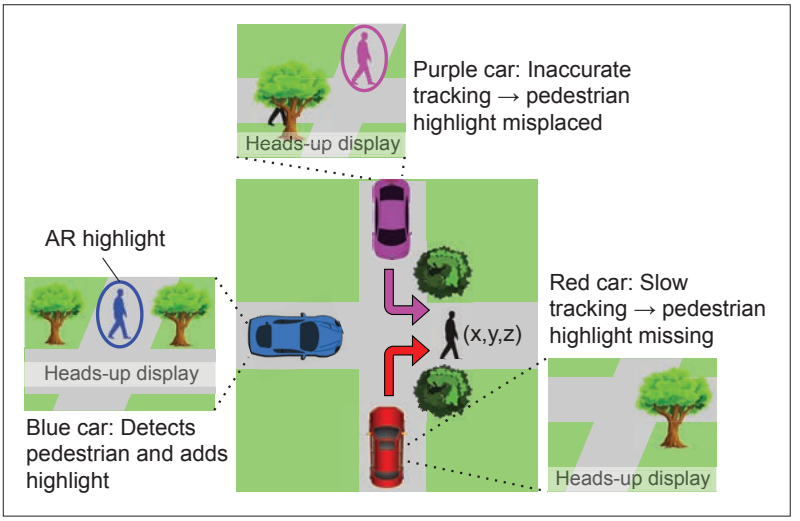

FIGURE 1. Example illustrating importance of accurate and fast localization.

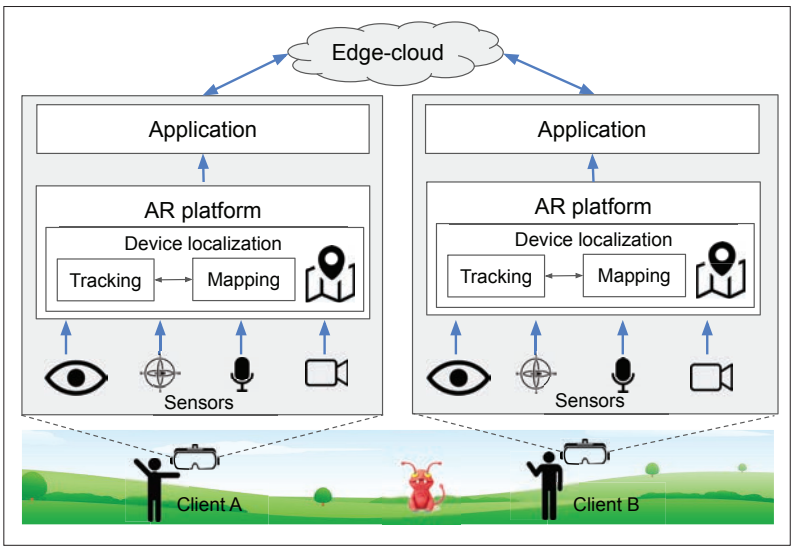

FIGURE 2. Multi-user AR workflow.

Single-User Localization

SLAM is a foundational algorithm used for localization in AR and typically involves two parallel steps, tracking and mapping. Tracking lets a client determine its pose (i.e., position coordinates and orientation) in the real world. Mapping creates a 3D representation of the world (i.e., a map) in which a client estimates its pose. The 3D representation has a coordinate system (called the world frame) with the origin at an arbitrary point in the real world. Tracking is relatively lightweight and should run in real-time; otherwise, localization and hence hologram rendering will be slow (as in the red car example above). Mapping runs concurrently with tracking, albeit at a slower time scale due to its computational complexity [4]. Tracking and mapping are related because tracking depends on updated maps for high accuracy (as in the purple car example above). If the environment changes rapidly or the user moves to a new area, fast mapping is more important. If the environment is relatively static (e.g., most objects in the real world do not move) or the user has low mobility and does not explore new areas,

The key differences between multi-user and single-user localization are the need for the localization process to bring together inputs from multiple users, merge them appropriately into the global map, and provide the relevant updates for individual users to localize within that global map.

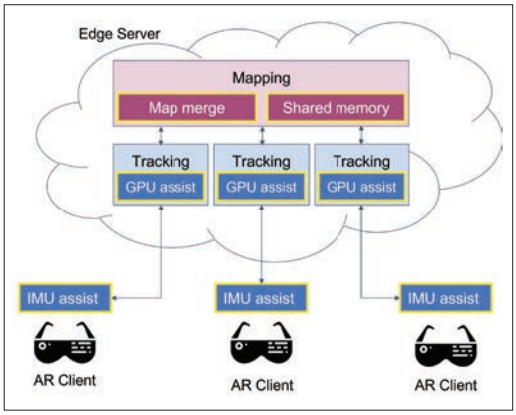

FIGURE 3. Edge-cloud centric architecture.

slower mapping is acceptable because the map does not need many updates. Next, we describe each of these steps in more detail.

The tracking module first decodes images (potentially extracted from video) obtained from the client's camera and extracts their image features. For example, ORB-SLAM3 [5] uses ORB (Oriented FAST and Rotated BRIEF); FAST (Features from Accelerated Segment Test) finds corner features in an image (by checking if a pixel's intensity differs greatly from it neighbors), while Rotated BRIEF are the descriptors for those found features. Around 1000 ORB features are extracted per image frame, which are then compared with the ORB features extracted from previous images of the environment in the SLAM map. By finding the common ORB features between newly processed images and the map, the location of the device can be estimated inside the map. The map is produced by devices who previously visited the physical area.

Mapping is triggered if the image read in by the tracking module is from a previously unseen location or contains informational cues (e.g., new features) for localization and tracking and therefore are not present in the local map, constructing a sparse point cloud. In that case, SLAM marks the frame as a "Keyframe." It next computes the poses of the Keyframe's features, known as "Mappoints," and inserts them into the map. Periodically, an optimization procedure, called bundle adjustment, runs to correct the pose error of each Mappoint. The information from tracking and mapping are stored in graph-based data structures and have temporal dependencies between frames [5].

Multi-User Localization

When multiple AR devices wish to place holograms in the real world, they first individually perform tracking and mapping as described above, resulting in individual maps (with individual world frames). They must then synchronize in order to create a consistent global map of the real world. This global map provides a common reference for the devices to describe the placement of the virtual objects,with similar data structures as the single-user case (except possibly larger due to contributions from multiple devices). Once the devices have a common global map and know the poses of the holograms, the holograms can be drawn on each device's display whenever they are in the client's field-of-view.

Next, we walk through the specific steps involved in merging the local maps of client A (which places a hologram), and client B (which receives and renders that hologram), when these computations are performed in the EC.

1. Client A uploads real-world visual data and hologram data (pose, geometry, textures) to a shared repository in the EC.
2. The EC server processes A's visual data using SLAM mapping to compute A's map.
3. Client B uploads its visual data about the real world to the EC server.
4. The EC server processes B's visual data into a map and with A's map to construct a global map. It returns B's pose in the global map.
5. Client B renders a hologram at the correct position and orientation on its display.

If there are more than 2 clients, the process is similar. A new client C would upload its data (step 3) and query the cloud for its pose in the global map (step 4). After receiving the query response, all clients continuously run tracking and mapping individually (regular SLAM), and synchronize with the EC server as needed, if there are updates to the global map.

The key differences between multi-user and single-user localization are the need for the above process to bring together inputs from multiple users, merge them appropriately into the global map, and provide the relevant updates for individual users to localize within that global map. Single user localization only employs its local map and does not require global map computation and coordination. Computing the global map (in step 4) can be computationally intensive due to the processing and combining of visual inputs from multiple users, which is challenging to perform locally on the end-points. Timely updates to the individual user devices is critical (in steps 1, 3, and 4) to ensure that the AR rendering is smooth and responsive. This requires a high bandwidth, low latency network.

Networked Architectures for Multi-user AR

In this section, we discuss network architectures to support SLAM for AR. We will outline three architectures starting from being most edge-cloud dependent to least: edge-cloud centric, edge-cloud assisted, and peer-to-peer. Each architecture makes different placement choices of where the merging of information from multiple users is performed, and what minimal updates have to be conveyed back to the users. These choices influence both the compute and communication requirements on the infrastructure.

Edge-Cloud Centric

A first architectural option to partition the work between the end-user devices and the edge cloud (EC) is to utilize the EC to have SLAM quickly and accurately perform tracking and mapping for multiple users. The EC-centric architecture helps supports the timely merging of multiple users' inputs and updates to the global map, as depicted in Fig. 3. The tracking and mapping components update themselves and communicate the "state" of the system (i.e., the shared merged map) to the clients. The system, which we call SLAM-Share [1], offloads tracking to the EC server, using the powerful compute capabilities that are likely avail-

able at the EC, such as GPUs and servers with multi-core CPUs and substantial memory. Specifically, a GPU can speed up two of the more significant complex processing tasks in SLAM's tracking: feature extraction and 3D point matching. For feature extraction, the key capability SLAM-Share introduces is the parallelization of FAST corner detection [5]. This is in contrast to a default approach of searching for the FAST features sequentially in each frame. SLAM tracking also requires the matching of newly obtained features with existing 3D points in the map. Unlike the default use of SLAM to sequentially match the features with points, SLAM-Share uses parallel threads in the GPU to match thousands of features and points concurrently, thus lowering the tracking time and making real-time tracking possible on the EC possible.

The EC-centric architecture relies on the network to transfer images (as a video stream) to the EC and determine a client device's pose. It is important to have accurate tracking results as discussed in earlier. Therefore, when there are transient higher network delays, SLAM-Share complements camera-based localization by utilizing the IMU to produce a pose at the client until the arrival of the pose from the EC. This is used only for short periods, and our evaluation shows that it causes less than 1 cm absolute trajectory error increment even when the round-trip network delay increases by 1 second.

SLAM-Share also improves the task of mapping by placing the global map in a shared memory buffer on the EC [6], which is accessible by a client mapping process corresponding to each user. The shared memory approach mitigates delays, serialization, de-serialization, data movement and other CPU overheads (because of zero-copy data transfers) and is very efficient for multi-user map merging. Each user's update is atomically applied to data structures in shared memory. Subsequently, a synchronized map merge of all the users' updates is performed. An example of these data structures and their processing is in [1]'s associated open-source code (https://github.com/network-lab2/slam-share) Each mapping process accesses the data directly in the shared memory buffer, and no inter-process communication time or conversion of the data structure is involved. Clients localize themselves based on this shared global map and update it instantaneously with minimal overhead using the data communicated by the clients.

Edge-Cloud Assisted

A second architectural option is EC-assisted [2, 7–9]. Edge-SLAM [2] and AdaptSLAM [9] are designed for single-user SLAM and perform tracking entirely on the user device, while some mapping is performed on the server. Such an architecture can be extended to a multi-user scenario [7, 8], which requires fast mapping and merging, as follows. Individual client devices perform tracking in this architecture, and each client then sends a map update to the server where the mapping is performed for each client individually. The maps of all the clients are then merged and the whole merged map is packetized and sent to each client device. Such an approach still requires substantial network bandwidth to transfer the complex, data-intensive SLAM maps between the server and the clients each time the maps are merged and updated, in contrast to the image/video upload in the EC-centric architecture. In Fig. 4, we compare the accuracy of an EC-centric architecture (SLAM-Share [1]) with an EC-assisted SLAM architecture (inspired by Edge-SLAM [2]). We examine the variation of the overall accuracy over time when the round-trip delay in the network is increased to 300 ms. For the EC-assisted architecture, this results in a significant increase in the Average Tracking Error (ATE), a common measure of localization error, in the interval 20-60 seconds in Fig. 4. This is due to delays incurred when the client waits for the large map update (comprising multiple megabytes) coming from the EC. Since the EC-centric architecture only requires the pose information (just a few bytes) from the server, its ATE does not change much because of the larger network delay. Thus the EC-centric architecture provides more accurate maps over the entire timeline.

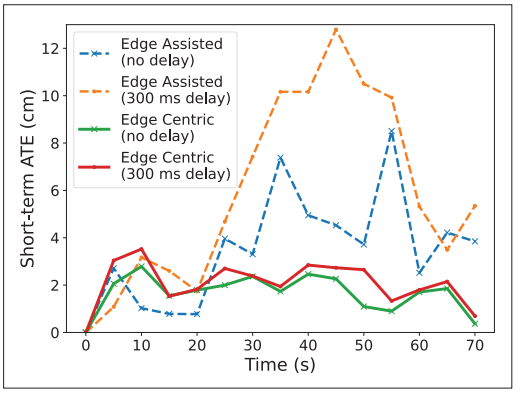

FIGURE 4. Short-term ATE: Effect of increased network delay — EC-assisted vs. EC-centric architectures.

The EC-centric architecture relies on the network to transfer images (as a video stream) to the EC and determine a client device's pose. It is important to have accurate tracking results as discussed in the Background section.

P2P

A final architectural option is to have clients perform their own tracking and mapping, and then share their results with each other, without the help of the EC [3]. The P2P option is only needed in a multi-user scenario. The map merge computation must be performed by the clients themselves. There are two design variants here. First, all clients can share information with each other and perform their own copies of the merge operation, along with tracking and mapping locally, as shown in Fig. 5. Alternatively, the clients can select one "primary" device that essentially acts as an EC (as in the EC-centric architecture), receiving information from other devices, performing the map merge, and distributing back the results. This primary device could be fixed for the duration of the AR session, or the primary role could rotate between devices. The key challenge in the multi-user scenario is the communication bandwidth and low latency needed to send maps back and forth between devices.

Use Cases

In this section, we discuss several potential AR use cases, their application-level requirements, and suitable network architectures to support these requirements. We selected these use cases because they represent distinct applications hav-

Self-driving vehicular systems and driver assistance systems need considerable resources to process the high bandwidth data generated by on-board cameras, light detection and ranging devices (LIDARs), radars, and/or other sensors needed to help the on-board inference engine with localization, object detection, and tracking.

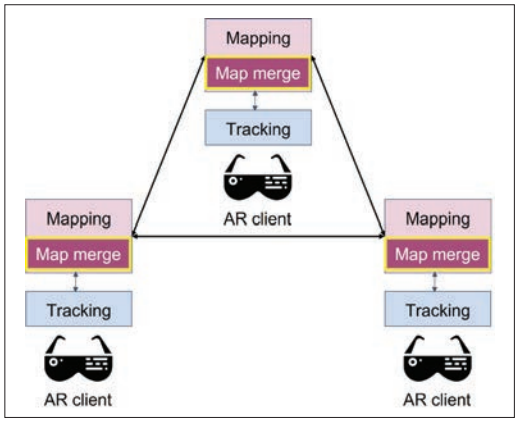

FIGURE 5. P2P architecture.

ing increasingly stringent requirements. Table 1 summarizes these use cases in terms of latency, spatial accuracy of the holograms, architecture, and expected user mobility. Latency refers to the end-to-end latency from when a triggering event occurs (e.g., a pedestrian walks into the road, or the user arrives at a location with a virtual creature) to when the hologram needs to appear on the user's own display, or that of another user. Spatial accuracy refers to the difference between where a hologram appears and where it is supposed to be. Safety critical applications such as autonomous vehicles and medicine (e.g., surgical training) have stringent latency requirements, with the latter likely to have equally critical requirements on spatial accuracy. Conversely, some relaxed entertainment applications do not have critical latency or accuracy requirements. We next describe three representative examples: entertainment, autonomous vehicles, and surgical training.

Collaboration and Entertainment

Different types of collaboration and entertainment AR applications exist. A *virtual whiteboard* that can facilitate collaborative work among users who are physically separated from each other [10]. Multiple users simultaneously collaborate on a virtual whiteboard, sharing its contents with users in different physical locations. *Community art* [10] allows users to create and view digital art within the same physical space and can also support public displays, allowing guests or passersby to view the artwork. The well-known *Pokemon Go* application enables multiple co-located users to view virtual creatures together in the same real-world environment. Finally, *paintball* [10] is a game where users aim and shoot virtual paint at virtual/real targets or other players.

Requirements: The latency and spatial accuracy requirements for the slower-paced games (virtual whiteboard, community art, Pokemon Go) can be fairly relaxed. Users can likely tolerate multiple seconds of latency until the holograms (a whiteboard, art pieces, or virtual creatures) appear or are updated. Some small deviation (centimeters to meters) in spatial positioning is also tolerable, as the functionality of the holograms is unlikely to be significantly affected by their positioning (e.g., a few centimeters of difference in the location of a large piece of virtual art would likely not affect user enjoyment). Note that the spatial accuracy discussed here is average real-time accuracy. If users experience a temporary reduction in accuracy (e.g., from loss of tracking), but regain improved accuracy shortly thereafter, it should be acceptable for such applications.

In contrast, fast-paced games like paintball would have tighter accuracy and latency requirements. Accurate calculations are necessary to determine whether holograms (e.g., paint) collide with the physical object (e.g., arm of a person). Therefore, spatial accuracy on the order of centimeters is required. Additionally, delayed feedback to players can cause users to wait for the results of each shot and negatively impact the overall game experience. Following similar analysis of the motion of holograms with real-world physics [11], the latency requirement on the AR system is several hundreds of milliseconds.

Supporting Network Architecture: For slower-paced games, where latency and accuracy requirements are looser, a P2P architecture may be suitable. Locally operating tracking and mapping can provide sufficient accuracy and latency to satisfy such applications' needs. P2P may also be particularly suitable if such games are played in areas without significant network or EC infrastructure. For example, community art could occur in a large park with poor cellular connectivity, lacking reliable access to the EC. However, for fast-paced games such as paintball with their stringent latency and accuracy requirements, the EC-centric architecture is vital to satisfy application requirements, since SLAM-based tracking and mapping can be slow on mobile devices [2]. A supporting argument for EC-centric SLAM is its ability to ensure consistency of the gameplay for all heterogeneous clients, since all computations are performed on the EC instead of being dependent on the hardware capabilities of individual clients.

Autonomous Vehicles

AR can be very valuable in vehicular scenarios to improve decision-making, whether for autonomous driving or to provide human driver assistance with advance warning. Obstacles (e.g., pedestrians) that appear in front of a vehicle may result in the vehicle taking evasive action, including sudden and severe braking or swerving to avoid the obstacle. Drivers of vehicles that follow behind can be helped by having a view of the obstacle on their own heads-up display early, based on the information provided by the vehicle in front, as shown in Fig. 1. Self-driving vehicular systems and driver assistance systems need considerable resources to process the high bandwidth data generated by on-board cameras, light detection and ranging devices (LIDARs), radars, and/or other 3D sensors needed to help the on-board inference engine with localization, object detection, and tracking. These can be enhanced by communicating the inference results to selected nearby vehicles (especially those behind the lead vehicle).

Requirements: There is a critical need to perform several of these tasks, such as localization, object detection and tracking, and rendering the AR image on the heads-up display quickly (e.g., less than 1 second, driving at highway speeds) [12, 13]. Providing timely, early warning to nearby drivers can help avoid accidents with the potential to save lives. Some spatial accuracy is required (e.g., several meters, within a lane), but cm-level precision isn't needed. Fluctuations in accuracy are unacceptable for autonomous driving, com-

pared to the games discussed earlier. A consistent and reliable level of accuracy is required for vehicular safety. Finally, the high mobility of autonomous vehicles increases communication challenges, especially with the need to meet tight latency requirements.

Supporting Network Architecture: EC-assisted SLAM can provide benefits for autonomous vehicles. The vehicles' high mobility means that they are constantly traveling to new areas, making fast map merging and alignment critical. For example, a vehicle that arrives at an intersection should quickly locate itself within the global map in order to learn about pedestrians in the area, as well as contribute information to the global map about places it has traveled from [12]. Merging these maps in a central location at the EC saves communication bandwidth compared to sending maps back and forth between vehicles. Given the desire for edge support, a question is whether EC-centric or EC-assisted architectures are appropriate. Since the vehicles generally have sufficient processing capabilities to run tracking and produce low-latency results on their own, an EC-assisted architecture may be suitable. However, in some cases, such as when the edge infrastructure is unavailable (e.g., if the vehicle is operating in a remote area), tracking can continue to run locally, and merging can fall back to local operation, that is, the P2P architecture. In other cases when communication latency is low and ultra-reliable, then an EC-centric architecture with tracking on the EC may more easily satisfy application requirements.

Surgical Training

AR is increasingly being used in medical scenarios, as it offers an intuitive and user-friendly approach that enables medical professionals to focus on solving medical problems. For instance, telementoring can integrate mentor annotations into the trainee's field of view using an AR display [14]. Artemis [15] is a AR-VR collaboration system that enables expert surgeons to work together with novice surgeons virtually, thereby enhancing the quality of patient care. The system allows experts to provide guidance by drawing annotations on a 3D reconstructed body of the patient, which can be calibrated to the novice surgeon's field of view. The novice surgeon can then follow the annotations to continue the surgical procedure. These systems typically operate in a training hospital.

Requirements: Surgical training requires high levels of spatial accuracy and very low latency, so as to not impair the performance of medical trainees, especially when multiple users are cooperating. Spatial alignment with mm-level accuracy between an expert's annotations and a trainee's actions is crucial for successful completion of the operation. Low latency is also crucial in order for the novice to see the expert's annotations in real-time. Mobility is low, since surgeons are likely stationary in an operating room, which avoids some of the more difficult protocol-level challenges to ensure low latency (e.g., during handovers in a cellular network).

Supporting Network Architecture: In hospital settings, since the AR experiences take place repeatedly in the same rooms, the rooms can be outfitted with additional sensors (e.g., 3D cameras) and combined with on-device sensors to improve accuracy. Thus, the AR experience can be supported by EC-centric or EC-assisted architectures. In particular, an EC-centric architecture may be preferred due to its ability to provide the highest accuracy and lowest latency, compared to alternative architectures. Furthermore, the EC provides a persistent repository for the global maps of these rooms. This means that the global maps can later be accessed by subsequent devices operating in the same room, reducing initialization latencies.

Use Case	Latency	Spatial accuracy	Architecture	Mobility
Pokemon Go, community art	seconds	meters	P2P	High (world-scale)
Whiteboard	sub-second	cm	P2P	Low (room-scale)
Paintball	sub-second	cm	EC-centric	Moderate (building-scale)
Autonomous vehicles	sub-second	meters	EC-assisted	High (world-scale)
Surgical training	10 ms	mm	EC-centric	Low (room-scale)

TABLE 1. Characterization of multi-user AR use cases.

Conclusions

A multi-user AR experience must provide users with a consistent view of the same holograms, for which devices must communicate and share computation results, requiring network support. This article outlined several edge-cloud-supported network architectures to assist AR devices and discussed their applicability across several representative use cases. For applications with less stringent requirements on latency and spatial accuracy, on-device computations and peer-to-peer communications may be suitable. For applications with stringent requirements, the heavy computing is preferably moved to the edge cloud to take advantage of its increased capabilities and a central point to share information. Meeting application requirements in this setting requires low latency and reliable communication links, as well as some on-device computation to provide approximate computation results in the event of communication delays and failures.

Acknowledgments

This work was supported by NSF grants 1817216 and 1942700.

References

[1] A. Dhakal *et al.*, "SLAM-Share: Visual Simultaneous Localization and Mapping for Real-Time Multi-User Augmented Reality," *Proc. ACM CoNEXT*, 2022.
[2] A. J B. Ali, Z. S. Hashemifar, and K. Dantu, "Edge-SLAM: Edge-Assisted Visual Simultaneous Localization and Mapping," *Proc. ACM MobiSys*, 2020.
[3] K. Apicharttrisorn *et al.*, "Breaking Edge Shackles: Infrastructure-Free Collaborative Mobile Augmented Reality," *ACM SenSys*, 2022.
[4] G. Klein and D/ Murray, "Parallel Tracking and Mapping for Small AR Workspaces," *Proc. IEEE ISMAR*, 2007.
[5] C. Campos *et al.*, "Orb-slam3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap Slam," *IEEE Trans. Robotics*, 2021.
[6] B. Schäling, *The Boost C++ Libraries*, 2011.
[7] P. Schmuck *et al.*, "Covins: Visual-Inertial Slam for Centralized Collaboration," *Proc. IEEE ISMAR*, 2021.
[8] J. Xu *et al.*, "Swarmmap: Scaling Up Real-Time Collaborative Visual Slam at the Edge," *Proc. USENIX NSDI*, 2022.
[9] Y. Chen, H. Inaltekin, and M. Gorlatova, "Adaptslam: Edge-Assisted Adaptive Slam With Resource Constraints via Uncertainty Minimization," *Proc. IEEE INFOCOM*, 2023.
[10] K. Ruth, T. Kohno, and F. Roesner, "Secure Multiuser Con-

tent Sharing for Augmented Reality Applications," *Proc. USENIX Security*, 2019.
[11] Z. Chen *et al.*, "An Empirical Study of Latency in an Emerging Class of Edge Computing Applications for Wearable Cognitive Assistance," *Proc. ACM/IEEE Symp. Edge Computing*, 2017.
[12] H. Qiu *et al.*, "AVR: Augmented Vehicular Reality," *Proc. ACM MobiSys*, 2018.
[13] R. Ravindran, A. Azgin, and K. K. Ramakrishnan, "Edge Transport (ETRA): Edge Transport Protocol Architecture for Next Generation Mobile IoT Systemds," *Proc. 2019 IEEE Globecom Workshops*, 2019, pp. 1–6.
[14] D. Andersen *et al.*, "Avoiding Focus Shifts in Surgical Telementoring Using an Augmented Reality Transparent Display," *Studies in Health Technology and Informatics*, vol. 220, 2016, pp. 9–14.
[15] D. Gasques *et al.*, "Artemis: A Collaborative Mixed-Reality System for Immersive Surgical Telementoring," *ACM CHI*, 2021.

Biographies

Jiasi Chen is an Associate Professor at the University of Michigan, and was previously at the University of California, Riverside. She received her Ph.D. from Princeton University and her B.S. from Columbia University. Her research interests include multimedia systems and mobile computing.

K. K. Ramakrishnan [LF] is a Distinguished Professor at the University of California, Riverside. Earlier, he was at Digital Equipment Corporation, and then AT&T Labs-Research. He is an ACM Fellow and AT&T Fellow. He has an M.E. from IISc, India, and MS, Ph.D. from the University of Maryland, College Park.

Aditya Dhakal received a Ph.D. in computer science from the University of California, Riverside and M.S. degree in computer science from the University of Connecticut. He is interested in GPUs, Systems for Machine Learning, interconnects for accelerators and augmented reality.

Xukan Ran received a Ph.D. in computer science from the University of California, Riverside and a B.S. degree from Xidian University. His research interests include edge computing and augmented reality on mobile devices.