Mixed Reality Collaboration between Human-Agent Teams

Thai Phan[*]
USC Institute for Creative Technologies

Wolfgang Hönig[†]
University of Southern California

Nora Ayanian[†]
University of Southern California

Abstract

Collaboration between two or more geographically dispersed teams has applications in research and training. In many cases specialized devices, such as robots, may need to be combined between the collaborating groups. However, it would be expensive or even impossible to collocate them at a single physical location. We describe the design of a mixed reality test bed which allows dispersed humans and physically embodied agents to collaborate within a single virtual environment. We demonstrate our approach using Unity's networking architecture as well as open source robot software and hardware. In our scenario, a total of 3 humans and 6 drones must move through a narrow doorway while avoiding collisions in the physical spaces as well as virtual space.

Index Terms: Human-centered computing—Virtual reality; Computer systems organization—Robotic autonomy

1 Introduction

Simulations can test complex systems before they are conceived, but most are not conducive towards multiuser collaboration. There is a need for a simulation test bed that allows multiple labs to conduct research cooperatively without the transportation of specialized equipment and resources between locations. We combine two ideas to simplify the collaboration of humans and physical entities, even when geographically dispersed: synthetic prototyping and mixed reality. Early synthetic prototyping [3] demonstrated that it is possible to use a game engine to play and interact with vehicle dynamics in a driving simulator, without the support of additional physical entities. Mixed reality has been used to test interactions between quadrotor drones and virtual humans [1]. We propose a mixed reality approach that uses the Unity 3D game engine to construct a virtual environment that combines multiple physical locations containing physically embodied entities such as humans and robots. Humans wear VR headsets to perceive other physical and virtual entities. We demonstrate our approach in a collaborative scenario, where humans, collocated in two physical locations, have to move through a narrow doorway. They are followed by autonomous drones and all entities must avoid physical and virtual obstacles while moving.

2 System Description

We consider two physical spaces, one with 2 users and 4 drones and the other with 1 user and 2 drones. One physical space has a physical door, while the other does not (see Fig. 1). Users in both physical spaces wear a Samsung Gear VR headset and are in the same virtual environment (VE) with the same door represented virtually. Each user has a pair of drones following in a simple line formation. The VE also contains virtual obstacles such as walls, human avatars, and representations of the drones. As each user passes through the open door, the drones must break formation in order to pass through also.

For motion tracking, one space uses PhaseSpace and the other uses VICON. The major system components are outlined in Fig. 2.

[*]e-mail: tphan@ict.usc.edu
[†]e-mail: {whoenig, ayanian}@usc.edu

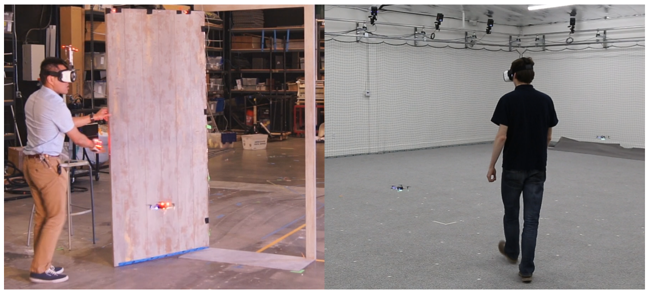
Figure 1: Left: Location A with physical door using PhaseSpace tracking. Right: Location B using VICON tracking.

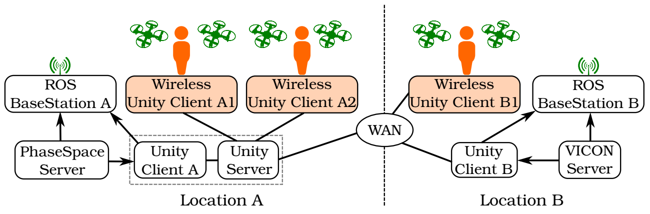

Figure 2: System diagram. Location A has 2 humans and 4 drones, while location B has 1 human and 2 drones. Humans and drones are tracked using PhaseSpace and VICON, respectively, and their poses are shared with the Unity Server.

We use Unity for the server and clients, for rendering, path finding, and state synchronization. We use the Crazyswarm stack [2] to control the flight of Bitcraze Crazyflie 2.0 quadcopters. Each of the two locations uses a computer (base station) running ROS (Robot Operating System). ROS BaseStation A commands 4 Crazyflies while BaseStation B commands 2 Crazyflies. All VR headsets act as Wireless Unity Clients which connect to the the Unity Server.

2.1 Network Design

An intracampus WAN allows our two tracking volumes to connect to each other with minimal number of hops. We implement a server-client architecture using the Unity game engine's High Level API (HLAPI). Unity also provides a Low Level API (LLAPI), both of which are built on top of a transport layer, which uses UDP packets[1]. The HLAPI suits our needs because it provides the core network components necessary to allow Unity clients to relay data to each other through the Unity server. One computer runs the Unity Server and Unity Client A in a single process. Unity Client A receives pose information of the PhaseSpace motion-tracked users and drones on its local LAN. It also receives pose information of all other users and drones via the Unity Server. Unity Client B receives pose information of the VICON motion-tracked users and drones on its LAN. It in turn receives pose information of the PhaseSpace-tracked users and drones via the Unity Server.

The Wireless Unity Clients only receive pose information for all users and drones in order to render their correct placement inside

[1]See https://docs.unity3d.com/Manual/UNetUsingTransport.html

the VE. They do not transmit user data or input to the Unity Server. Both Unity Client A and B process the pose information locally. The Unity Server itself does not process the pose information as it simply updates all clients with the most current information.

For closed-loop control, Unity Client A sends a new goal position for each drone (at a rate of 10 Hz) via a Python TCP socket connection to ROS BaseStation A. Likewise, Unity Client B sends goal data to ROS BaseStation B. Both base stations communicate and command drones via a custom RF radio on the 2.4 GHz band and not through 802.11 WiFi. Wireless Unity Clients communicate on IEEE 802.11ac to reduce congestion on the 2.4 GHz band.

2.2 Motion Tracking

All of the Gear VR headsets are outfitted with unique marker arrangements for 6DoF tracking; using PhaseSpace active LED markers or VICON passive retro-reflective markers. The 4 drones tracked by PhaseSpace are modified to carry a PhaseSpace microdriver with 2 tracking LEDs, drawing 3.7 V from the drone's battery. Unity Client A only receives partial pose data from these drones – their yaw and position. The 2 drones tracked by VICON are outfitted with small spherical markers adhered directly to the frame. The marker arrangements are unique, so Unity Client B receives full 6DoF pose data. Unity Client A uses the PhaseSpace SDK, while Unity Client B connects to VICON's integrated VRPN server. Data is unbuffered and only the most recent data is used.

2.3 Drone System

The Crazyflie 2.0 hardware and software is open-source. Using the Crazyswarm architecture [2], the majority of in-flight computation is done on the Crazyflie's 32-bit, 168 MHz ARM microcontroller. The two ROS base stations transmit the pose and goal positions to the drones over 2.4 GHz radio, at a broadcast rate of 100 Hz and 10 Hz, respectively. An onboard Extended Kalman Filter (EKF) fuses pose data from the tracking system with data from the Crazyflie's IMU. Trajectory planning and control is done on-board.

2.4 Agent Navigation

Unity Clients A & B calculate the paths for the drones by representing each one as an agent inside the virtual environment. Unity's navigation system uses the A* search algorithm[2]. This provides suitable reactive planning to avoid collisions with users, obstacles, and other agents. It also simplifies the maneuvers of the drones for the purpose of flying in close proximity with humans, each having an invisible 0.5 m radius boundary against collisions.

Unity generates a navigation mesh (NavMesh) for the environment in which paths for agents will be calculated on a 2D plane. Paths change as dynamic obstacles such as users, the door, and other agents cause obstructions. When an agent has no available path to reach a goal position, the agent stops moving until an obstruction is moved or a new path becomes available.

Unity Clients A & B will have identical local copies of the NavMesh, but will calculate paths independent of each other. If one of the client's network connection is interrupted, the other client will calculate paths using the most recent pose information reported by the Unity server. No client-side prediction is performed. Finally, invisible boundaries surrounding the VE prevent drones in both tracking volumes from flying into untrackable areas.

2.5 Mixed Reality Design

The physical door (2040 mm $\times$ 920 mm) is also tracked with PhaseSpace markers (see Fig. 1). As the door opens and closes, its virtual representation moves accordingly. One user wears a hand strap with PhaseSpace markers. When this user reaches for the doorknob, others see his/her avatar reach accordingly inside the VE. We are also using an off-the-shelf solution for inverse kinematics (by RootMotion[3]) to animate avatars as they locomote. A screenshot of the VE (as rendered in the Unity Clients) is shown in Fig. 3.

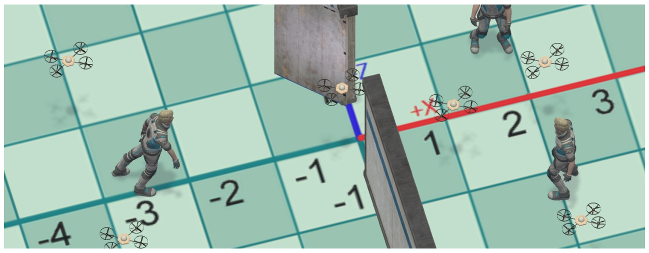

Figure 3: Overhead view of 3 users and 6 agents in total, sharing the same virtual environment.

The Wireless Unity Clients run on Samsung Galaxy S7 phones. Each client renders a stereoscopic view without distortion correction using a simplified rendering pipeline to prevent the phones from overheating. In order to give users improved spatial awareness, a HUD was designed to present them with an overhead view of their surroundings, indicating the location of other agents and users.

3 Discussion and Future Work

Unity has been the de facto rendering engine for VR research applications in recent years. We have shown that its multiplayer networking and AI architecture can facilitate mixed reality collaboration between human-agent teams. Using our approach, a wide variety of physical spaces can be interconnected for collaboration. Humans can work safely within their own physical confines, while their intelligent counterparts can operate in more hazardous environments. This also enables the testing of heterogeneous human-agent teams. For example, UAVs might operate in a facility equipped to simulate air turbulence, while anthropomorphic robots might operate in a mock disaster zone. Our approach is also well suited for mixed reality prototyping since we will be able to substitute networking and AI components with alternative implementations. Path finding can be undertaken by an external system or onboard the drones themselves. Optical motion tracking can gradually shift towards decentralized localization using onboard sensors. In these cases, peer-to-peer networking can better simulate intercommunication between drones. We believe that our test bed will help the development of human-agent interactions and will serve to acclimate end users with autonomous systems.

Acknowledgments

This work was sponsored, in whole or in part, by the U.S. Army Research Laboratory (ARL) under contract number W911F-14-D-0005. Statements, expressed opinions, and content included do not necessarily reflect the position or the policy of the United States Government, and no official endorsement should be inferred.

References

[1] W. Hönig, C. Milanes, L. Scaria, T. Phan, M. Bolas, and N. Ayanian. Mixed reality for robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5382–5387, 2015.

[2] J. A. Preiss*, W. Hönig*, G. S. Sukhatme, and N. Ayanian. Crazyswarm: A large nano-quadcopter swarm. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3299–3304, 2017. Software available at https://github.com/USC-ACTLab/crazyswarm.

[3] R. Spicer, E. Evangelista, R. New, J. Campbell, T. Richmond, C. McGroarty, and B. Vogt. Innovation and Rapid Evolutionary Design by Virtual Doing: Understanding Early Synthetic Prototyping. In *Proceeding of Simulation Interoperability Workshop*, 2015.

[2]See https://docs.unity3d.com/Manual/nav-InnerWorkings.html

[3]See http://root-motion.com/