Cloud-based Multimodal Human-Robot Interaction Simulator Utilizing ROS and Unity Frameworks

Yoshiaki Mizuchi¹ and Tetsunari Inamura¹,²

Abstract—A new software architecture for a simulator of human-robot interaction (HRI) in virtual reality (VR) environments is proposed. Since collecting and storing a massive amount of data concerning multimodal interaction experiences is an important task concerning research on HRI, a cloud-based VR platform, named “SIGVerse,” which reduces costs of developing real robots and interaction experiments in the real world, is proposed. The reusability of virtual robot software is restricted in a real environment due to difference between VR and real robot architectures; therefore, a new architecture utilizing the Unity and ROS frameworks is proposed. The proposed architecture provides functionalities for constructing scalable 3D environments, embodied and social interaction via the Internet, compatible robot software, high-fidelity sensor feedback, and recording/playback of interaction. To demonstrate the feasibility of the proposed architecture, the performance of SIGVerse in terms of simulating multimodal information in actual interaction applications was evaluated, and the latency between avatars synchronized via cloud computing was measured. Additionally, interactive behaviors of robots and avatars in a VR environment and a real environment were experimentally compared, and the comparison results confirm that the VR behaviors of robots and avatars were almost the same as the behavior of a robot in a real environment.

I. INTRODUCTION

In regard to robot intelligence, a function that can learn from real experience (learning function, hereafter) is one important factor. Acquisition of motion skills [1] and learning by demonstration [2] using multimodal information have been proposed as learning functions for the real world. Simple learning such as imitation of trajectory and dynamics of a sensorimotor map have been achieved by real robots; however, several problems arise when the robot attempts to acquire higher cognitive knowledge when performing an activity in real daily life. To model such a daily life activity, not only simple sensor information acquired by the robot, but also a history of interaction between the robot and humans which includes embodied physical motion, utterances of humans, and emotional and cognitive status of humans is required. Storing such embodied and social information concerning daily-life activities is a difficult task for conventional robot systems due to the huge cost of experiments and maintenance of real hardware [3][4].

One of the standard solutions to reduce the cost of experiments is to use simulations. So far, many robot-simulation systems for a variety of applications have been proposed [5][8][9][10][11]. However, in the case of conventional simulation systems, simulating human-robot interaction in a simulation environment involves another problem, namely, modeling a human activity as a reproducible computation model. Additionally, it is necessary to recruit research subjects (i.e., “humans”) to participate in a laboratory experiments with robots. The cost of such human participation remains a major problem concerning standard simulation systems. One approach to avoid these problems is to utilize cloud computing and virtual-reality (VR) systems.

A VR system named SIGVerse [12], with which humans can login with an avatar and interact with virtual robots, has been proposed by Inamura et al. Since this system is server-client based, several participants can join an experimental environment on a server from separate client computers. However, this system suffers problems such as a lack of reusability of robot software due to the original robot-control API/SDK and difficulties in developing device-driver modules for many kinds of VR interfaces.

Aiming to solve those problems, a new system architecture for SIGVerse is proposed in this paper. Unity middle-ware is adopted for the VR application. Since Unity is adopted, it is not necessary to write device drivers for each VR interface. Additionally, a connection mechanism to bind Unity and the ROS environment was developed. This new version of SIGVerse (Ver.3) enables users to develop robot software in ROS and put a virtual robot model into VR applications based on Unity. Thanks to the proposed system architecture, the cost of software development can be significantly reduced. In the following sections, the new architecture is explained in detail, and the evaluated performance of SIGVerse (Ver.3) in realizing human-robot interaction applications in VR environments is presented. Additionally, the future direction of application of the proposed system is discussed in terms of storage of huge data concerning cognitive experiences in daily-life activities and use of that experience in cloud-based robotics.

II. REQUIREMENTS CONCERNING A PLATFORM FOR COLLECTING DATA ON MULTIMODAL INTERACTION

The aim of this study was to collect and leverage multimodal-interaction-experience data collected in daily-life environments that require embodied social interaction. For example, collaborative cooking tasks, dialogue-management...
systems dealing with vague utterances, and gestures and gazing behaviors are assumed as the target situations involving such interaction. In these situations, a robot must observe and learn social behavior of the humans it is interacting with, and it must solve ambiguities based on past interaction experience. In such complex environments, the robot should collect the following multimodal data:

1. Physical motion/gestures during interaction (including gaze information)
2. Visual information (i.e., what image the agents see)
3. Spatial information (i.e., position of agents and objects)
4. Voice interaction (i.e., utterance of agents)

Additionally, the following functions must be provided.

1. User login to avatars in the VR environment
2. Multiple-user login to the same VR server through the Internet at the same time
3. Recording and replaying time-series multimodal interaction data
4. Attaching control programs of real robots to virtual robots

As mentioned in the introduction, several simulation systems already realize the functions listed in Table I. However, none of them realize functions (i) to (iv) above at the same time.

Our previous system (SIGVerse ver.2 [12]) has been utilized for studies such as analysis of human behavior [14], [15], learning of spatial concepts [16], [17], and VR-based rehabilitation [18]. Those studies used multimodal data (1) to (4) above and functions (i) to (iii); however, the reusability of conventional SIGVerse is restricted due to its API. Therefore, the system needs to keep functions (i) to (iii) and realize function (iv). A software architecture that realizes the above functions is proposed in the next section.

### III. ARCHITECTURE OF SIGVERSE

The detailed architecture of SIGVerse (ver.3), including a participant and a robot, is shown in Fig. 1. SIGVerse is a server/client system based on Unity’s built-in networking technology. The server and clients have the same scene composed of 3D object models such as avatars, robots, and furniture. By communicating information of registered objects via the Internet, it is possible to synchronize events in each scene.

The participant can login to an avatar via VR interfaces such as head-mounted displays (HMDs), motion capture devices, and audio headsets. According to the input from such VR devices, behavior of the participant is reflected on the avatar by Unity scripts. Perceptual information such as perspective visual feedback is provided to the participant. Thus, the participant can interact with the virtual environment in a similar manner to a real environment.

The proposed VR simulation system has a bridging mechanism between ROS and Unity. Software for virtual robot control can be reused in real robots without modification, and vice versa.

The information for reproducing multimodal interaction experiences is stored on a cloud server as a large-scale dataset of embodied and social information. By sharing such information, users can reproduce and analyze the multimodal interaction after the experiment.

#### A. Construction of VR environment

To store the embodied and social information collected from daily-life activities, large-scale and complex 3D environments must be constructed. Unity provides a flexible and sophisticated user interface for building VR environments and interaction experiences. Consequently, users can easily construct large-scale complex environments and share them. Additionally, thousands of ready-made 3D models and scripts (known as assets) are shared in the Unity Asset Store. For high-fidelity sensing feedback, Unity provides high fidelity and real-time rendering and depth-data generation.

#### B. VR interfaces

To perform multimodal interaction in the VR environment, participants should login to an avatar using immersive VR interfaces. Recently, a variety of VR interfaces such as VR headsets (e.g., HTC Vive, Oculus Rift+Touch, and FOVE) and motion-capture devices (e.g., Kinect, Leap Motion, and Perception Neuron), have been developed. The appropriate such device to use is selected according to the purpose of the interaction experience. However, any such interface can be easily utilized with the proposed system without having to write device drivers because most are supported by Unity’s assets. As shown in Fig. 2, by utilizing Unity’s ready-made assets, it is easily to perform multimodal interaction tasks by using such VR devices.
C. Server/client configuration

Unity’s built-in server-client functionality was utilized for developing online multiplayer games. The Unity server and clients are generated from the same project. Users execute it as a server, a client, or a host (both server and client) by selecting a mode. For this reason, the server and client have the same scene and objects. Dynamic objects such as avatars or robots are spawned by the server when the client logs in to the shared environment. As well as the server, each client can have authority over objects. Accordingly, a local player can interact with objects (e.g., grasping) in real-time without delay due to communication with the server. Information concerning such an authorized object is transmitted to the server and broadcasted to all clients. Thus, avatars, robots, and objects can interact with each other via the network.

D. Compatible robot software

The compatibility of robot software is a crucial factor for robotic simulators. Recently, most real robots and their applications are developed with robot middleware such as ROS and OpenRTM. Particularly, ROS has emerged as a standard robotic middleware owing to its modularity and a number of software libraries. An essential factor of robot middleware is the management of data communication among plural processes. ROS-based software is basically modeled as a group of asynchronous processes (known as nodes). These processes communicate information such as sensory feedback, commands, and states via message passing. By supporting the interface for such message passing between Unity and ROS, existing ROS-based resources for real robots can be reused for virtual robots.

E. Mechanism for connecting ROS and Unity

To control a robot in a VR environment, sensory feedback and robot commands should be passed between Unity scripts and ROS nodes. The most important factor in realizing the integration of ROS and Unity is the communication protocol between them. Software systems for bridging ROS and Unity have been proposed by Hu et al. [22] and Downey et al. [23]. As for those systems, motor commands and sensor information are transferred basically using rosbridge. However, if users attempt to send a massive amount of sensor information such as camera images from Unity, a bottleneck on transfer speed is created. Previous works [22][23] did not discuss how
to transfer camera images in realtime; accordingly, a new technique to realize realtime transfer based on the BSON format through a TCP/IP connection is proposed in the following.

As a ROS functionality, a rosbridge framework provides a JSON API and a WebSocket server for communicating between ROS and external non-ROS programs. JSON is a text-based data exchange format that represents pairs of keywords and values. Although the rosbridge protocol covers sending and receiving ROS messages, its performance for parsing large JSON data such as images is insufficient to satisfy real-time sensor feedback. For this reason, a specific server (sigverse_rosbridge_server) for communicating large data volumes was implemented. For speeding up communication, the BSON format was employed instead of JSON. BSON is a binary-encoded serialization with a JSON-like format. The use of BSON offers following advantages: communication data size is reduced to less than that of text-based data, a conversion process between text and binary is not required, and data is represented as key-value pairs compatible with ROS messages. When ROS messages are advertised by Unity scripts, the main thread of the sigverse_rosbridge_server generates a new thread for each topic as a ROS node. Each thread receives ROS messages from the Unity scripts and publishes them in ROS core as ROS topic messages.

F. Recording and playback of multimodal interaction

The multimodal interaction experiences should be reproduced from the stored data; therefore, storing only sensory information of agents is not enough to reproduce such situations with multimodal information that is the same level as when participants demonstrated the interaction. The proposed architecture can reconstruct such interaction experiences including sensory information by simulating the phenomena in the VR environment, even if active objects such as avatars are in motion.

IV. EVALUATION OF PERFORMANCE OF PROPOSED PLATFORM

A. Virtual sensors

To demonstrate that sensor feedback based on the proposed architecture satisfies the real-time requirement, the performance of virtual sensors of the proposed platform was evaluated. Virtual RGB and depth sensor information is generated and communicated with a host PC and a virtual machine. The specification of the host PCs and virtual machine used for the evaluation is listed in Table II.

Screen shot images in Unity and ROS are shown in Fig. 3. As shown in the upper image, an avatar stand in front of a virtual turtlebot with an RGB-D sensor. RGB image data and depth data are correctly received as messages on the ROS side, as shown in the lower screen in Fig. 3. Both images have 640×480 pixels resolution. Each pixel of the RGB image is 24-bit (8 bit × 3 channels) format. Each pixel of the depth image is 16-bits format. Accordingly, data size of an RGB image is 900 [kB], and the data size of a depth image is 600 [kB].

To evaluate the validity of the proposed architecture, the frequencies of the sensor feedback by WebSocket communication with JSON format and Socket communication with BSON format were measured. In this evaluation, a laptop PC and a high-power desktop computer were measured. Average frequencies of RGB image data and depth data depending on the computer and data format used are listed in Table III. Frequency of JSON communication is less than 1.0 [fps], even if one high-end PC was used. JSON format represents image-buffer data as a string array. The conversion between binary data to string data and the subsequent spitting process

### Table II

<table>
<thead>
<tr>
<th>Specification of Host PCs and Virtual Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PC #1</strong></td>
</tr>
<tr>
<td>CPU</td>
</tr>
<tr>
<td>Number of processors</td>
</tr>
<tr>
<td>RAM</td>
</tr>
<tr>
<td>GPU</td>
</tr>
<tr>
<td>Number of cores (virtual machine)</td>
</tr>
<tr>
<td>RAM (virtual machine)</td>
</tr>
</tbody>
</table>

### Table III

<table>
<thead>
<tr>
<th>Frequencies of Virtual RGB-D Data Depending on Protocols and PCs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PC #1</strong></td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>PC #2</strong></td>
</tr>
</tbody>
</table>
TABLE IV  
RESULT OF NETWORK-TRAFFIC EVALUATION.

<table>
<thead>
<tr>
<th>Number of clients</th>
<th>Server</th>
<th>Client #1</th>
<th>Client #2</th>
<th>Client #3</th>
<th>Client #4</th>
<th>Client #5</th>
<th>Client #6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Received data ((d_r)) [bytes/s]</td>
<td>Transmitted data ((d_t)) [bytes/s]</td>
<td>Received data [bytes/s]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5,160</td>
<td>8,077</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9,919</td>
<td>32,096</td>
<td>16,049</td>
<td>16,045</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>14,686</td>
<td>63,088</td>
<td>21,022</td>
<td>21,025</td>
<td>21,044</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>19,448</td>
<td>117,875</td>
<td>29,474</td>
<td>29,437</td>
<td>29,479</td>
<td>29,437</td>
<td>29,479</td>
</tr>
<tr>
<td>5</td>
<td>24,225</td>
<td>187,953</td>
<td>37,614</td>
<td>37,566</td>
<td>37,620</td>
<td>37,566</td>
<td>37,562</td>
</tr>
<tr>
<td>6</td>
<td>28,918</td>
<td>261,194</td>
<td>43,543</td>
<td>43,496</td>
<td>43,539</td>
<td>43,496</td>
<td>43,561</td>
</tr>
</tbody>
</table>

B. Latency between a local avatar and other avatars

To evaluate the performance of the server-client configuration, we measured latency between a local avatar and other avatars whose motion are synchronized via the Internet was measured. The configuration of the measurement is shown in Fig. 4. Several laptop computers were set up as a server, a publisher for motion data, and clients. Axis Neuron Software in the publisher computer broadcasted pre-recorded motion data through a TCP/IP connection. All the client computers and the motion publisher computer were in the same place and directly connected via wired LAN. The motions of local avatars were synchronized according to the received motion data. Each client computer was also connected with a remote server computer via the Internet. Information concerning the pose of the local avatars was sent to the server at a frequency of 10 frames per second. In this manner, the poses of the non-local avatars in each Unity client were synchronized with the poses of the corresponding avatars in the Unity server.

The parameters for synchronizing the motion of avatars are listed in Table V. In this experiment, motions of 24 joints were synchronized at 10 frames per second. Each joint has nine variables for three degree of freedom (DoF) for position, three DoF for rotation, and three DoF for scale. Each variable is a 4-byte float.

Motions of avatars in a client were measured and the latency between a local avatar and non-local avatars when two clients, four clients, and eight clients were logged in to the server, respectively, were evaluated. The actual distance between the server computer and clients computers was approximately 33.5 km.

Screenshots at almost the same moment in time when the (two, four, or eight) clients were in the motion are shown in Fig. 5. The colored avatar is the local avatar, and the white avatars are the non-local avatars. Relatively speedy motion data included in the Axis Neuron Software was chosen as examples. It is clear from the figure, that although the motions of non-local avatars are slightly delayed, the motions of the avatars are synchronized via the Internet.

Angle of the right wrist during a part of the motion is...
where \( j \) is number of joints, \( f \) is frequency of synchronization between client avatars, \( v \) is the number of parameters to define joint status, \( b \) is data size for each parameter, and \( r \) is data reduction rate. Each joint is synchronized only when the change of the joint movement exceeds the threshold. Joint data are compressed for communication that should be used in Unity. Although we had chosen a motion of Tai Chi which consists of continuous whole body motion, data size was reduced approximately 43% in this experiment. The server sent to each clients approximately data to each client at 8 KiB/s per avatar. Size of data transmitted from server \( d_t \) is calculated as

\[
d_t = n \sum_{i=0}^{n} (d_i + \alpha),
\]

where \( \alpha \) is the size of additional data considered to be data for synchronizing each identical joint between the server and clients. In this experiment, \( \alpha \) was approximately 2,600 bytes per avatar. According to the above equations, under these experimental conditions, \( d_t \) is calculated as 28.9 KiB and \( d_t \) as 264.5 KiB for six clients. These values almost correspond to the measured sizes of communicatied data. The measured data size is slightly smaller than the calculated values because frame rate slightly lowers depending on the number of synchronized objects in a scene. In the case that 30 clients log in to a server under the same condition, \( d_t \) will be 1.1 Mibps, \( d_t \) will be 51.7 Mibps, and each client will receive 1.7 Mibps data. This network traffic is sufficiently feasible for interaction between multiple avatars and robots via the Internet. Although it depends on the specification of the CPU, network, and number of synchronized joints, this performance demonstrates that dozens or so articulated avatars such as human and robot can log in to the remote VR environment and interact with each other at the same time.

C. Application of proposed platform

To verify whether a robot can behave in a virtual environment by reusing the controller software for real robots, the following human tasks were demonstrated by a turtlebot mobile robot. In this demonstration, based on turtlebot_follower included in the ROS package for turtlebot, a robot follows a walking human by centering its rotation and keeping the distance to the closest blob in a depth image. Behaviors of the turtlebot and the human in a real environment and a virtual environment are compared. Motion of the human was recorded by a motion capture device (Perception Neuron), and the recorded motion data were sent to an avatar.

The behaviors of the human and robot are shown in Fig. 7. As shown in each figure, the avatar walks and the virtual turtlebot could follow it reasonably well. Vertical position \( x \) and horizontal position \( z \) of the avatars in both environments are shown in Figs. 8(a) and (b), respectively. In these figures, the cross-points denote the positions of the real human, and the line shows the positions of the avatar in virtual environment. As shown in the figures, the behavior of the avatar almost correspond to the behavior of the real human in the real environment. The behaviors of the real
robot and the virtual robot, are shown in Figs. 9(a) and (b), respectively. The behavior of virtual robot almost correspond to that of the real robot. In Fig. 8(b), the horizontal position of the avatar is slightly displaced after 25 seconds. This displacement is due to the accumulated error of inertial sensors on Perception Neuron. According to the behavior of the avatar, the horizontal position of the virtual robot is also displaced after 25 seconds. These results show that the virtual robot can behave in almost the same manner in the virtual environment as in the real environment.

V. CONCLUSION

Aiming to solve two key problems concerning robotics simulation (i.e., limitation on reuseability of robot-control software and cost of developing VR interface module), a new system architecture for SIGVerse was proposed. In the architecture, Unity middleware is adopted for VR applications and ROS is adopted for developing of robot software. A bridging mechanism between Unity and ROS enables users to develop human-robot interaction applications in an immersive VR environment in shorter time. Several basic functions
of the proposed system were evaluated, and the evaluation results confirm that the current performance of SIGVerse with the proposed architecture satisfies that required by basic applications.

REFERENCES