Connecting the Real World with the Virtual World - Controlling AIBO through Second Life

Evan Wong[1], Wei Liu[1,⋆], and Xiaoping Chen[2]

[1] School of Computer Science and Software Engineering
University of Western Australia, Crawley, WA 6009
[2] Department of Computer Science
University of Science and Technology China, Anhui, China
wei@csse.uwa.edu.au

Abstract. The main aim of this project is to develop middleware so that the Second Life online virtual space (virtual world) can be used to simulate and control the movements of a Sony AIBO robot (real world) in a wireless environment. This paper details the design of an immersive teleoperation system, and the rationale behind the design. The prototype proves that the concept of teleoperation with greater sense of immersion is achievable and can lead to future work in application domains such as smart home and immersive remote operating machinery in the industry such as mining.

1 Introduction

The Internet, as a communication backbone of modern society is further exploited, in the past decade, in the scenarios of *teleoperation*[1], to ensure the safety of personnel in high-risk industries such as mining, aerospace and defence. Broadly speaking, teleoperation means controlling and operating a device remotely by an operator from afar.

Benali et al. [1] identified that teleoperation over the Internet faces major challenges due to the fact that there is no guaranteed upper bound to the potential large time delay and the consequent data loss. Therefore, in the past, researchers are mainly focusing on addressing the network delay issues. Benali et al. [1] proposed a system with a network analyser. It measures the packet round-trip time for evaluating the quality of service, and employs a control mode manager to make decisions on whether to continue a given task or not. Xue et al. [2] attempts to guarantee a stable data stream by using a handshaking protocol between the server and the client. Both Xue et al. and Benali et al. concluded that the time delay can be managed and resolved through either human intervention [1] or using a fuzzy controller triggered only by a sensor event [2].

Therefore, this project takes the manual control approach to counter the effect of the network delay. This allows us to centrate on the idea of providing

⋆ Corresponding Author.

[1] Telelabs Project: http://telerobot.mech.uwa.edu.au/information.html

J. Baltes et al. (Eds.): RoboCup 2009, LNAI 5949, pp. 402–413, 2010.

a higher level immersion for what we termed as *immersive teleoperation*. It is widely agreed that, comparing to the traditional human computer interface with buttons and drop-down menus, being able to operate machinery in an immersive 3D virtual environment will enhance job satisfaction, which is especially attractive to trainees.

The overall aim of this project is to use a popular and accessible 3D virtual online environment - Second Life as a medium to teleoperate the Sony AIBO (short for **A**rtificial **I**ntelligence ro**BO**t), more specifically the Sony AIBO ERS-7 robot. Second Life (SL), created by Linden Lab, is a 3D online virtual environment that attempts to model the surface of an Earth-like world in a reasonably life-like way [3]. Users create models or avatars, which are essentially the "people" of the virtual world. The Sony AIBO is an artificial intelligence robotic pet dog designed and manufactured by Sony. The AIBO is an autonomous agent able to gain information about the environment and make decisions without human intervention [4]. The AIBO also has a built-in wireless adapter receiving and transmitting data wirelessly [5]. The joints as well as the vision, acoustic and motion sensors can be accessed directly via the OPEN-R platform provided by Sony [6] through programming in C++. This allows the AIBO to be programmed to perform specific behaviours. The design objective of this project is that users can control the movements of the AIBO in a physical environment through moving an avatar in Second Life.

This paper reports the design and development of such a system with technical details on how to intercept Second Life packets and how the different modules in the system communicate. We hope as a preliminary system, this paper can offer some insights and starting points for similar projects focusing on remote machinery operations via immersive virtual environment. Section 2 provides a general overview of existing architectures adopted by the state of the art teleoperation systems. In Section 3, we propose the design and development of the system with detailed discussions on the roles of each module and how they are implemented and communicate with each other. Section 4 presents experiments results. The paper concludes with an outlook to future work in Section 5.

2 Related Work

2.1 General System Architecture

The system architectures widely adopted are variations on the Internet-based client-server structure [2,7,8]. The control architectures typically comprise of a *central main controller* sitting on the server that talks to *embedded controllers* within the robot. For example, the architecture by Bensoussan et al. [9] uses two controllers; a central controller on the server computer and a real-time controller which sits on the robot (in the context of their paper, the robot is a vehicle). The central controller manages the sensors, camera, ultra-sound, radio communication; the real-time controller in the robot, on the other hand, controls of locomotion of the vehicle. Benali et al. [1] and Xue et al. [2] designed the controllers as classical fuzzy controllers using fuzzy logic. Hohl et al. [10] take this

one step further by introducing the notion of cross-compilation of the controller so that the robot can run independently of the host computer once the main controller is compiled.

To enable teleoperation by human operators, a common practice is to use a graphical user interface (GUI) that sits on the client computer as an desktop or a Web application. This GUI is usually coded as a Java applet with VRML (Virtual Reality Modelling Language). VRML is a language that can be used to visualise and build virtual worlds which include 3D objects, light sources, animations, and user representation via avatars [7]. This allows for manual control of both the simulated and the real robots. The idea is that the real robot will mimic whatever that is shown on the screen by having the GUI communicating with the controllers on the server and the robot. This mode of manual control is what we adopted for this project. However, it can be easily extended to various levels of control, including, semi-automatic, fully automatic [2], and a hybrid of all three [1]. It is also noted that it is possible to use multiple hosts simulating and controlling the robot concurrently [8].

2.2 Second Life Networking Architecture

Second Life implements a client-server architecture. Separate servers are employed to handle different tasks, including Login Server, Data Server, Map Server and etc. Among these, the servers of interest to us are the *simulators*.

The world of Second Life is made up of many *simulators*. Simulators are basically servers that simulate a 256x256 metre region each. When the avatar in Second Life moves through the Second Life world, it moves from one simulator to another [11]. The simulator keeps track of where everything is and sends the location of objects in Second Life to the client (a.k.a. viewer). The simulator is in charge of running the physics engine in the Second Life world and it does collision detection as well.

The *viewer*, on the other hand, does not handle any collision detection. It sends velocities and simple physics information to the simulator, keeping track of avatar movements. When collisions occur, updates are sent from the simulator to the viewer, which is then updated accordingly on the viewer [11].

All these communication amongst the simulators as well as the communication between the simulators and viewers are done via UDP through *circuits*. A circuit is basically a two-way UDP connection between two nodes.

Packets transmitted on Second Life between the client and the server have a consistent layout. A packet is divided into three parts: the header, body and the appended acks. The *header* contains information regarding the packet itself [12]. The *body* of a Second Life packet contains the message number, which is a numeric encoding of the message types, followed by the actual message data [12]. For example, the message storing information about the movement of the avatar is an `AgentUpdate` message. Finally, the packet might have acknowledgements appended. Acknowledgments from previous reliable messages "piggyback" and fill up as much of the packet as it can fit [12].

3 System Design and Development

3.1 System Overview

As show in Figure 1, our design follows the widely adopted client-server architecture discussed in Section 2. The GUI is, in this case, the Second Life client installed on a client computer. The only difference is that the central main controller (the VRInterface in Figure 1) resides in the client computer, rather than a separate server machine. This is because the Linden Lab's Second Life simulator servers are dedicated and close source, unlike the systems discussed in the literature, which have servers located close the robot to minimise extra network delay. The embedded real-time controller we choose to use is the URBI platform, which is loaded to the programmable memory stick on AIBO. Detailed discussions of each module are available in the following subsections.

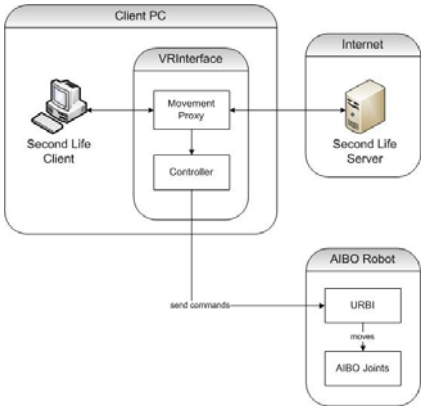


Fig. 1. System Overview

3.2 Virtual Real Interface - VRInterface

The core module of our system that enables intercepting Second Life packets and sending commends to the robot is the `VRInterface`. It consists of two submodules, the `MovementProxy` and the `Controller`.

Movement Proxy is an application we developed based on a third party software library: the `SLProxy` of the `libsecondlife` libraries. It sits in between the Second Life client and the Second Life servers, analysing the packets that are being transmitted along the stream.

`SLProxy` is a library that allows applications to act as a proxy between the Second Life client and the servers. `SLProxy` tracks the circuits, modifying the sequence numbers and acknowledgments when changes are made to the packet stream from a third-party application that makes use of the `SLProxy` library [13]. Therefore, applications using the `SLProxy` library can read, inspect and modify any packet that is transmitted and received between the Second Life client and the servers. It can also remove packets or inject new packets into the packet stream [13].

The Movement Proxy shown in Figure 1 only reads the packets but theoretically, based on the capabilities of the `SLProxy` library, it can also be used to inject packets back into the Second Life client. Injecting packets can be used to realise scenarios such that the Second Life avatar moves to resemble the autonomous movement of the AIBO in the real environment.

Movement in Second Life is handled by both the client and the servers. The client sends packets which contains information of the movements of the avatar

to the servers up to 20 times per second [14]. The packet that we are interested in is the `AgentUpdate` message [2]. The `AgentUpdate` message contains information from the camera in the Second Life viewer which is sent to the simulator [15] at a rate of up to 20 times per second. The `ControlFlags` variable inside the `AgentUpdate` message contains information about the movements of an avatar in Second Life. Each movement is given a constant integer which is used to shift 0x1 by the given constant. As an example, for a movement in the forward (positive) direction,

```
const U32 CONTROL_AT_POS_INDEX  = 0;
const U32 AGENT_CONTROL_AT_POS  = 0x1 << CONTROL_AT_POS_INDEX;
```

The `ControlFlags` variable will contain `0001`, which is the decimal 1.

However, a "nudge" is sent by the client first, followed by the normal key press shown in the example above. This "nudge" is to impart velocity when brief keypresses are made by the client [14]. For a "nudge" in the forward direction,

```
const U32 CONTROL_NUDGE_AT_POS_INDEX  = 19;
const U32 AGENT_CONTROL_NUDGE_AT_POS  = 0x1 << CONTROL_NUDGE_AT_POS_INDEX;
```

The `ControlFlags` variable will contain `1000 0000 0000 0000 0000`, which is the decimal 524288.

If the forward key continues to be held down after the "nudge", Second Life will store a combination of the forward movement with a "fast" movement (the avatar is walking faster) into the `ControlFlags` variable.

```
const U32 CONTROL_AT_POS_INDEX  = 0;
const U32 CONTROL_FAST_AT_INDEX = 10;
const U32 AGENT_CONTROL_AT_POS  = 0x1 << CONTROL_AT_POS_INDEX;
const U32 AGENT_CONTROL_FAST_AT = 0x1 << CONTROL_FAST_AT_INDEX;
```

The `ControlFlags` variable will now contain a combination of the forward movement and the "fast" movement `0100 0000 0001`, which is the decimal 1025.

What this allows for is that a combination of keyboard presses stored and transmitted within one packet. If, for example, the avatar makes a diagonal movement (forward and left), the two movements can be combined, which means less packets are required to be transmitted, thus reducing any unnecessary delays from repeatedly sending a packet to move one step forward, followed by a packet to turn left, and then yet another packet to move one step forward, yet another to turn left, and so on.

When a packet containing the movement information is sent to the corresponding server, the server computes the current position and transmits the position back to the client. As mentioned in Section 2.2, the client does not perform collision detection and it is the job of the simulator or server to do that task. The position of the avatar seen on the viewer is therefore velocity and acceleration interpolated [14]. This means that if there is a long delay in the network, the avatar in the Second Life viewer may appear to unrealistically walk through an obstacle such as a wall. The position of the avatar is then again

[2] For a template of the `AgentUpdate` message please see [15].

unrealistically "corrected" at the time when the viewer eventually receives the packets containing correct position calculated by the server, after long network delays. One will see that the avatar is pushed back (after walking through a wall) to reflect that there is an obstacle in front of it.

The **Controller**'s job is to analyse the packets from the Movement Proxy and translate the movements of the avatar in Second Life stored in the `ControlFlags` variable into commands that is understood by the AIBO robot.

To translate the movements of the avatar in the `ControlFlags` variable into commands understood by the AIBO robot, a lookup table is used to map the `ControlFlags` variable into URBI commands for the AIBO robot. URBI (Universal Real-Time Behavior Interface) is a software platform by Gostai supporting development of robotics and AI applications [16]. It is chosen in preference to other platforms such as Tekkotsu because it is a universal platform that works with not only the Sony AIBO robots, but also a variety of other robots, independent of operating systems and programming languages [17].

We implement this lookup table as a hash map data structure, in which the keys would be the possible values (unsigned 32-bit integers) that the `ControlFlags` variable might hold, while the values of the hash map would be the URBI commands the robot understands.

Table 1. Hash Map

Key	Value
1025	walk.go(1)
1026	walk.go(-1)
256	walk.turn(30)
512	walk.turn(-30)

Commands sent to the AIBO robot will be repeated and continuous if the three latest packets received in a row contain the same `ControlFlags` variable. If a new movement is made, the `ControlFlags` variable of the new packet will not be the same as the `ControlFlags` variable of the last two packets, forming the stopping rule for the robot. A stop command is then sent to the robot followed by commands for the new movement.

In order to send commands to the AIBO through URBI, a new socket connection is established at start-up between the Controller and the AIBO robot on port 54000, which is the port for URBI commands on the robot. TCP is used as the protocol for the connection between the Controller and the AIBO robot due to its more reliable nature as compared to UDP.

3.3 Creating a Dog-Like Avatar in Second Life

The default avatar in Second Life is a basic human avatar. In order to create a closer representation of the AIBO robot in the Second Life virtual environment, a dog-like avatar is created. Objects in Second Life are created from basic primitive objects such as a cube, cylinder, prism, pyramid etc. These objects can be transformed by stretching, shrinking and then put together to form a larger object. Using the primitive objects provided in Second Life, objects such as arms, legs, chest, head and so on can be created and then attached onto the avatar to form a dog-like avatar, thereby creating a closer representation of the AIBO robot in Second Life.

3.4 Communications between the Modules

Figure 2 illustrates the sequence of events and interactions that occur between the different services when the Movement Proxy is started up and connections are established with the Second Life server and the Sony AIBO robot.

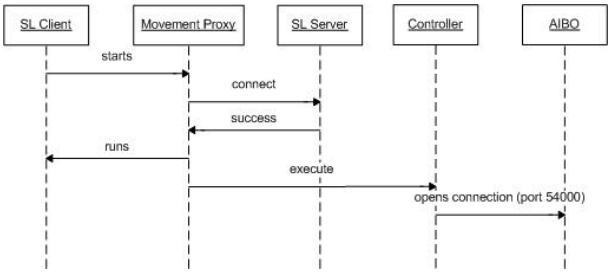


Fig. 2. Sequence of Events at Startup

The sequence of events and interactions between the services during runtime is quite similar to those during execution with slight differences being that the services in the system perform different actions.

4 Experiments and Results

4.1 Experiment Environment and Settings

The experimental environment was kept stable by ensuring the following parameter stayed as constant as possible throughout the experiments:

The speed of the Internet and networks cannot remain constant and fluctuates based on the amount of traffic on the network. Long network delay can result in lag and potential loss of packets. Therefore, in cases when the Internet speed drops to a level deemed unsuitable, the experiments are abandoned until speeds are improved and deemed suitable again.

Table 2. Experimental Environment

Robot:	Sony AIBO ERS-7
Robot Platform:	URBI
Workstation:	Intel Core 2 Duo 2.20Ghz processor
	2GB of memory
	NVIDIA GeForce 8400M GS video card
Workstation Platform:	Windows XP Professional
Second Life Version:	Release 1.19.0(5)
Wireless Network Protocol:	802.11b
Internet Speed:	$>$ 1.5Mbps

4.2 Setting Up The Movement Proxy

To allow the Movement Proxy to capture and analyse data being transmitted between the client and the server, the client is set to connect to the server through the Movement Proxy, which is on the same workstation as the Second Life client through port 8080. An extra flag is added to the Second Life client to connect to the Second Life server through the Movement Proxy. This can be done by appending an `loginuri` when running a Second Life client:

```
C:\Program Files\SecondLife\SecondLife.exe -loginuri http://localhost:8080
```

This ensures that before running the Second Life client executable, the Movement Proxy is loaded up first. When the Movement Proxy is loaded up, it establishes a connection to the Second Life server. At the same time the Controller, which is compiled and executed together with Movement Proxy as part of the same executable file, opens a connection to the AIBO robot through URBI on port 54000. When everything is loaded up and the connections are established successfully, the AIBO robot will make a sound and the Movement Proxy will display that it has been loaded up successfully as shown in Figure 3.

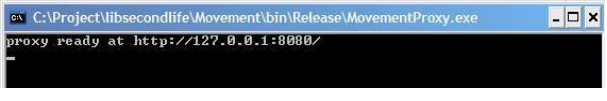


Fig. 3. Movement Proxy

4.3 Movement Testing and Observations

To test the success of the system in terms of achieving the main objective of being able to teleoperate the AIBO robot in the real world and mirror the movements of its virtual representation in Second Life, basic movements were performed on an avatar in Second Life and results were observed on the AIBO robot. These basic movements performed are: Moving Forwards (Figure 4) and Backwards (Figure 5) in a straight line; Turning 90° to the Left (Figure 6) and to the Right (Figure 7).

The corresponding figures of each of the four basic movements mentioned above show frame-by-frame shots of the AIBO robot performing the movements with the Second Life avatar moving in the Second Life client visible in the background.

As part of the experiments to test the forward and backward movements, markers were placed on Second Life that were 10.0 Second Life units apart in a straight line (Y-axis). Markers were also placed some distance apart to form a rectangular region in the real world. The purpose of placing the markers in Second Life and the real world was to measure the closeness in the representation of the movements of the AIBO in the real world with that in the Second Life virtual environment. The coordinates of the markers in Second Life were:

Marker 1 X: 140.0 Y: 90.0 Z: 27.157
Marker 2 X: 140.0 Y: 100.0 Z: 27.157

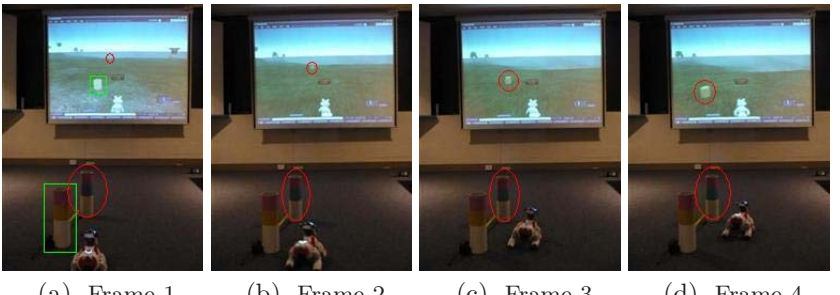
The Z coordinate represents the distance of the marker from sea level in Second Life. Both the X and Z coordinates were unchanged while the difference between the Y coordinates between markers 1 and 2 was 10.0. A distance of 10.0 units in Second Life reflects a distance of approximately 0.6m in the real world.

As one can observe from Figure 4, while the avatar is moving closer to the identified marker in Second Life, the AIBO robot is also moving closer. In Figure 5, while the avatar is moving away from the identified marker in Second Life, the AIBO robot is also moving away from the markers. Based on the observations, the straight line movements (moving forwards and backwards) of the AIBO robot mimicked that of the avatar in Second Life very closely. This also includes faster and slower walking movements in a straight line.

In Figure 6 and Figure 7, using the identified stable object in Second Life, we can say that turning movements performed by the AIBO robot did mimic the turning movements made on the avatar in Second Life. However, it was not mimicked as close as we would have liked them to be. This might be due to the lack of detailed and fine calibrations of the turning angles in Second Life and in the real world. The representation of the AIBO robot as a 2-legged avatar and the fact that the AIBO is a 4-legged robot had some negative impact on accurately calibrating and representing the movements of the avatar in Second Life for the AIBO robot. If the bounding box of the avatar is proportional to the size of the robot in the real world, a much closer representation of the AIBO robot in Second Life can be achieved.

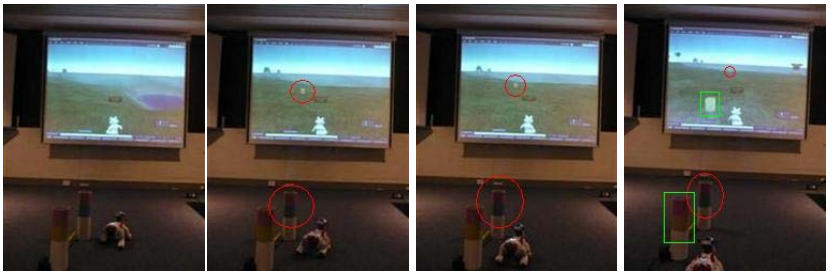
More complex movements such as the movement of individual joints and movement of the head were not performed due to the limitation of Second Life in creating more sophisticated avatar. This is further discussed in Section 5.

As stated in Section 4.1, the Internet speed was monitored to ensure that it is fast enough (> 1.5Mbps) so that the delays would not have a great impact on the experiments. However, given the dynamic nature of wireless networks and the Internet, occasional delays on the network imply that the latency cannot always stay constant. The experiments were aborted if the Internet speed is deemed to



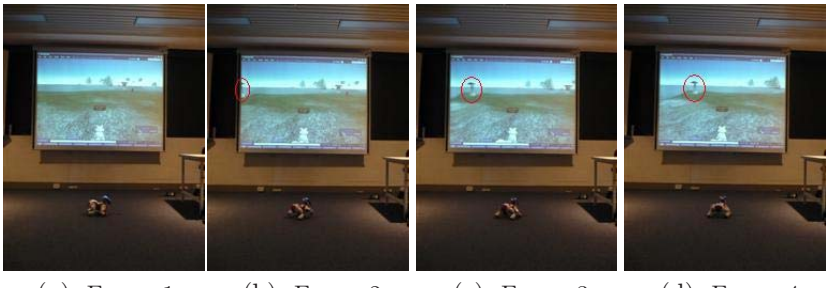
(a). Frame 1 (b). Frame 2 (c). Frame 3 (d). Frame 4

Fig. 4. Forward Movement - Walking Forward in a Straight Line



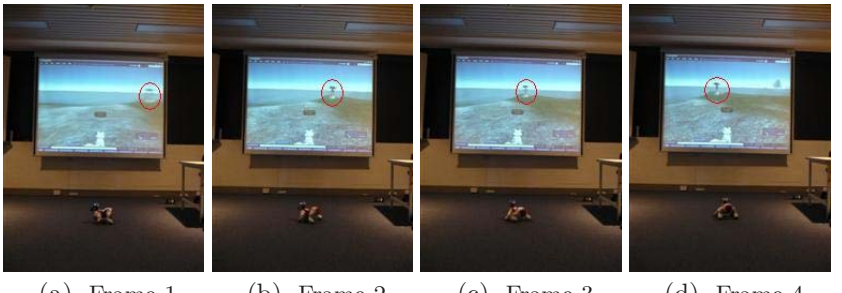
(a). Frame 1 (b). Frame 2 (c). Frame 3 (d). Frame 4

Fig. 5. Backward Movement - Walking Backward in a Straight Line



(a). Frame 1 (b). Frame 2 (c). Frame 3 (d). Frame 4

Fig. 6. Left Turn Movement - Turning 90° to the Left



(a). Frame 1 (b). Frame 2 (c). Frame 3 (d). Frame 4

Fig. 7. Right Turn Movement - Turning 90° to the Right

be unacceptable to ensure that the Internet speed does not greatly affect the outcome of the experiments.

We measured that the average time delay from the time the avatar moves in Second Life and when the AIBO robot responds and starts moving is in a range of 500 milliseconds to 3 seconds. This range was obtained by sending ICMP packets ("pinging") to the Second Life server and the AIBO robot from

the client workstation, in which the Second Life client as well as the Movement Proxy and Controller resides. The round trip time in which the packets were sent out by the client and when the response is received is averaged and recorded. This procedure was repeated 20 times during different times of the day to take into account the variation of network traffic during the course of a day.

The impact of delays on the system can be minimised by the use of a buffer that temporarily stores a set number of packets at a given time and then filtering the packets out to ensure that re-sent packets that were delayed do not interfere with the current packet stream that is being received, processed and sent to the `Controller` by the client.

5 Conclusions and Future Work

This project is a software engineering exercise to connect the real world (the Sony AIBO robot) with the virtual world (the Second Life virtual environment) in which movements made by the avatar in Second Life were to be mirrored in the real world through the Sony AIBO robot.

Based on the experiments and the results obtained, the main objective of the project has been met. The delay as noted above is within an acceptable range of 500 milliseconds to 3 seconds. This range could be made smaller and improved in the future. There are other 3D virtual environment servers (open source) available which can be installed locally or within an intranet to significantly reduce the delay. A buffer can also be implemented to ensure that delays are taken into consideration.

The prototype system demonstrates that immersively teleoperating robots through the Second Life virtual environment is feasible. Immersion for the operator is the key attractor of this type of 3D virtual environment. For example, a mining site could be replicated within Second Life and machines could be teleoperated through the Second Life environment. There is also growing trend in Second Life where houses are built to replicate the real houses of individuals. Through immersive teleoperation, users can control robots in their homes to carry out various tasks remotely.

In future work, we are planning to import an 3D model of AIBO into a 3D environment for a precise representation such that individual joints of the robot could also be controlled. This will extend the basic movements reported here to complex ones require the coordination of body parts such as sitting and dancing. We are also interested in feeding packets into Second Life so that the robot's autonomous movements in real life are mirrored on Second Life. The preliminary testing has been carried out successfully by injecting short chat messages through the proxy which is then shown in Second Life.

Acknowledgement. This work is supported by Australian Academy of Science and Chinese Academy of Science through the Scientific Visiting Awards, and the University of Western Australia Research Grant Scheme 2008.

References

1. Benali, A., Wasiak, V., Fontaine, J.G.: Remote robot teleoperation via internet. a first approach. In: Wasiak, V. (ed.) Proceedings. 10th IEEE International Workshop on Robot and Human Interactive Communication, 2001, pp. 306–312 (2001)
2. Xue, X., Yang, S.X., Meng, M.Q.H.: Remote sensing and teleoperation of a mobile robot via the internet. In: Yang, S.X. (ed.) 2005 IEEE International Conference on Information Acquisition, p. 6 (2005)
3. Ondrejka, C.R.: A piece of place: Modeling the digital on the real in second life. SSRN eLibrary (2004)
4. Flake, G.: The Computational Beauty of Nature: Computer explorations of fractals, chaos, complex systems, and adaptation. MIT Press, Cambridge (2000)
5. Sony Entertainment Robot Europe: Aibo brochure eng 2004 (2004), `http://support.sony-europe.com/AIBO/downloads/en/brochure_04lr_en.pdf`
6. Serra, F., Baillie, J.C.: Aibo programming using OPEN-R SDK Tutorial. ENSTA (2003)
7. Hoyer, H., Jochheim, A., Rohrig, C., Bischoff, A.: A multiuser virtual-reality environment for a tele-operated laboratory. IEEE Transactions on Education 47(1), 121–126 (2004)
8. Puente, S.T., Torres, F., Ortiz, F., Candelas, F.A.: Remote robot execution through www simulation. In: Torres, F. (ed.) Proceedings of 15th International Conference on Pattern Recognition, 2000, vol. 4, pp. 503–506 (2000)
9. Bensoussan, S., Parent, M.: Computer-aided teleoperation of an urban vehicle. In: Parent, M. (ed.) ICAR 1997. Proceedings of 8th International Conference on Advanced Robotics, 1997, pp. 787–792 (1997)
10. Hohl, L., Tellez, R., Michel, O., Ijspeert, A.J.: Aibo and webots: Simulation, wireless remote control and controller transfer. Robotics and Autonomous Systems 54(6), 472–485 (2006)
11. Second Life Wiki: Server architecture (2008), `http://wiki.secondlife.com/wiki/Server_architecture`
12. Second Life Wiki: Packet layout (2008), `http://wiki.secondlife.com/wiki/Packet_Layout`
13. axial: Slproxy - libsecondlife (2008), `http://www.libsecondlife.org/wiki/SLProxy`
14. Second Life Wiki: How movement works (2008), `http://wiki.secondlife.com/wiki/How_movement_works`
15. libsecondlife: Linden lab development message templates (2008), `http://www.libsecondlife.org/template/release/1.19.1.4.txt`
16. Wikipedia: Urbi (2008), `http://en.wikipedia.org/wiki/URBI`
17. Gostai: The urbi platform (2008), `http://www.gostai.com/urbi.html`