© 2013 Operational Research Society Ltd. All rights reserved. 1747-7778/13
www.palgrave-journals.com/jos/

A modular architecture for an interactive real-time simulation and training environment for satellite on-orbit servicing

Robin Wolff[1]*, Carsten Preusche[2] and Andreas Gerndt[1]

[1]Simulation and Software Technology, German Aerospace Center (DLR), Braunschweig, Germany; [2]Institute of Robotics and Mechatronics, German Aerospace Center (DLR), Oberpfaffenhofen-Wessling, Germany

Maintaining or repairing satellites in orbit is a delicate task that requires expert skills. The planning, training and analysis of on-orbit servicing (OOS) missions performed by astronauts or through remote operation using a robot is often time consuming and costly. Virtual Reality (VR) enables simulation and training in a flexible and safe environment. This paper describes an interactive real-time environment that supports a number of OOS tasks within an immersive VR environment. The system simulates the dynamic and kinematic behaviour of satellite components and provides photo-realistic visualization of satellite parts and the space environment. It integrates user interaction with haptic force feedback through a bi-manual haptic human machine interface, as well as simulates and interfaces to a humanoid robot for tele-operation.In order to provide a realistic experience at interactive frame rates, we propose a distributed system architecture, where the load of computing the physics simulation, haptic feedback and visualization of the complex scene is transferred to dedicated machines. The modular architecture is designed to allow the inclusion of further simulation processes. Several mechanisms for reducing the communication traffic have been implemented. This paper gives an overview of the system architecture, outlines the software implementation and documents an evaluation of the real-time performance of our system in detail. We describe how system performance was measured in terms of simulation timings and distribution load, as well as report on latencies at several stages. Results show that our distributed system is capable of providing visual and haptic feedback at high frame rates required for user interaction with end-to-end latencies of less than 8 ms and 3 ms, respectively.
Journal of Simulation advance online publication, 10 May 2013; doi:10.1057/jos.2013.10

Keywords: man-machine systems; training; distribution

1. Introduction

On-orbit servicing (OOS) is an inter-disciplinary field with increasing importance for the space industry. Although engineers take extensive care when designing and launching satellites, system failures do happen. A survey by Ellery *et al* (2008) showed that most failures occur during orbit injection or in the first year of operation, for example when folding out the solar panel. As space systems are costly, any failure is damaging, not only for commercial operators, but also for research missions. Today, almost no satellite is launched without insurance to cover for failure. However, not only failure, but also a limited lifetime and an increasing amount of space debris are motivations for OOS. With a growing number of satellites orbiting Earth, there is a growing need for innovative methods to repair failures, to refuel satellites for extending its lifetime, or to remove disabled and adrift satellites and other unused parts from orbit in a controlled and cost-effective way.

A well known example of OOS is the series of NASA Hubble Space Telescope repair missions, starting in 1993, where astronauts fixed and replaced parts in several extra vehicle activities (EVAs). However, manned missions like this are both expensive and put a high risk to the astronauts working outside the spacecraft. With the advances in the robotic field, robotic servicing constitutes an attractive alternative. The Canada arm on the International Space Station (ISS), for example, has frequently demonstrated successful operation (Currie and Peacock, 2002). Further ongoing projects and concepts for robotic OOS studies include the Deutsche Orbitale Servicing Mission (Rupp *et al*, 2009) with the goal to demonstrate the capturing of an uncooperative spacecraft; and the Orbital Life Extension Vehicle (Sellmaier *et al*, 2010), a commercial project with the aim to dock with communication satellites and take over the attitude and orbit control system for extending the satellite's lifetime. Some service robots work in automatic operation modes, either pre-programmed or (semi)-autonomous. However, n complex maintenance tasks or in situations where the

**Correspondence: Robin Wolff, Simulation and Software Technology, German Aerospace Center (DLR), Software for Space Systems and Interactive Visualization, Lilienthalplatz 7, Brunswick, Lower Saxony 38018, Germany.*
E-mail: robin.wolff@dlr.de

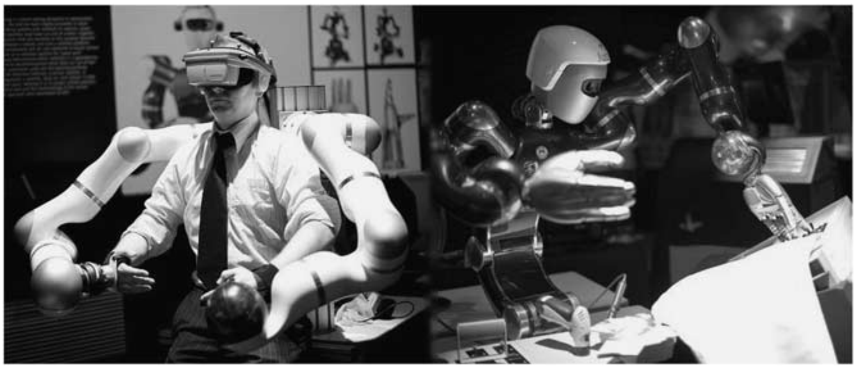

Figure 1 Demonstration of robotic on-orbit servicing using a tele-presence interface (left) to control a humanoid robot and manipulate a physical satellite mock-up (right).

cause of the failure is unknown and an investigation is required, the actions cannot be pre-programmed. In such cases, it is necessary to manually program the service robot in sequences or to operate the robot directly via a tele-operation interface. The German Aerospace Center (DLR) already started in 1993 to study ways and the effects of operating a robot in space from a ground station in the Robot Technology Experiment (Hirzinger *et al*, 1994), and later in the Robotic Components Verification on the ISS (ROKVISS) project (Landzettel *et al*, 2006). In ROKVISS, for the first time a force feedback joystick has been integrated to the tele-operation control of a robot in space.

Controlling a robot in complex scenarios is not trivial and usually requires a robot expert or a well-trained person. However, analysing and repairing a failure in a satellite can only be done by a satellite expert. Hence, for planning and performing appropriate repair tasks both satellite and robot experts have to work closely hand in hand. DLR developed a human-scale bi-manual haptic interface (Hulin *et al*, 2008), enabling the intuitive operation of a humanoid servicing robot (Ott *et al*, 2006) in a way where the user is able to control the robot's head and hands with the own head and hands, while seeing the same view as the robot sees through a head-mounted display (HMD), displaying the live video of the stereo-camera built into the robot's head. The combination of the bi-manual haptic interface and the humanoid servicing robot creates a tele-presence interface, with which a satellite expert can control a robot and intuitively perform servicing tasks without the direct help of a robot expert. The setup was used at the International Aerospace Exhibition – ILA 2010 in Berlin to demonstrate tele-operation of a service robot in a number of servicing tasks using a physical test satellite mock-up (see Figure 1).

Training, programming and testing the tele-operation of robots for OOS tasks may be done in physically based test beds, such as in Artigas *et al* (2006) or that shown in Figure 1. This has the advantage that specific hardware and control software to be used during the mission can be integrated directly into the test environment. However, certain components or environmental effects have to be simulated, which can make the setup complex. Moreover, testing in a physical environment induces the risk of damage to equipment or even could harm humans. In addition, making changes to the mock-up is often time consuming and connected with costs. Virtual reality (VR) offers a cost-effective alternative for simulation within a flexible and safe environment, and has already been used for simulation and training in many application areas, such as surgery or crane operation training.

This paper presents VR-OOS, a VR environment for OOS. Our goal is to provide a multi-modal virtual environment that can be used as a platform for the analysis, training and programming of OOS tasks, as well as to help to develop and examine new designs of serviceable satellites and servicing robots. The environment will be used to train astronauts in manned servicing missions, as well as to program, and eventually to tele-operate, service robots in space in unmanned missions. The challenges are to provide the accurate real-time simulation of the dynamics of the satellite components under real conditions, combined with photo-realistic high-resolution rendering of detailed virtual objects and environmental effects within an immersive, haptics-enabled virtual environment. The paper presents our approach to implement the underlying system of the interactive real-time simulation environment using a distributed architecture and an evaluation of its performance.

The paper is organized as follows: the next section outlines selected OOS scenarios that form the minimum set our system is expected to support. Then, an overview of the involved system components and their function is given. We then summarize the real-time challenges put on the system and describe the proposed system architecture. This is followed by a description of measurements for evaluating our system and a report of our results. Finally, we close the paper with a summary and an outlook for future work.

2. OOS scenarios

The repair and maintenance of satellites usually consists of a number of predefined sub-tasks that must be executed in a specific order. The goal of our proposed simulation environment is to train the procedure and correct sequence of actions within various OOS tasks. In order to support the training of a wide range of possible servicing scenarios, the system should provide a set of basic sub-tasks that often occur and can be combined to various complex servicing scenarios. Therefore, we selected a number of tasks that would occur in most servicing scenarios based on common EVAs:

2.1. Remove multi-layer insulation (MLI)

As satellites are usually covered by a MLI foil, the first action would be to open the MLI in order to reach the satellite components underneath. In most cases, the foil will have to be cut using a knife, scissors or other tools and then pulled open. Some satellites have their MLI foil attached by Velcro-like fasteners, which can be opened and closed again.

2.2. Loosen and tightening screws

Many parts, such as modules and covers, are fixed with screws. Thus, a second task is to loosen screws, and after the main work tighten them again. This is usually done using a cordless screwdriver.

2.3. Replace a module

A common task will be to replace electronic parts by exchanging a module. This will be done using a handle that is inserted into a module like a bayonet catch. After rotating the inserted handle by 90 degrees, the module can be pulled out. Inserting a module is done in the same way.

2.4. Toggle a switch

Before removing a module, a task will be to switch off the electronics on the module, and after inserting a new module to switch it on. This is done via a normal lever switch.

2.5. Take measurements

Finally, a common task is to take measurements at electronic parts. This task is represented using a digital voltmeter and touching specific measurement contacts with the measuring tip.

A milestone of the project is to demonstrate the execution of these tasks within the virtual environment and compare it to interacting with a physical mock-up. For this evaluation, the virtual mock-up, as seen in Figure 2, resembles the physical mock-up, seen in the foreground in the right image in Figure 1, in size and arrangement of objects.

The scenarios described above comprise the minimum set of tasks that are planned for the first prototype. The final version will have to allow the simulation of arbitrary tasks, possibly involving the handling of much more complex mechanisms, such as handling bayonet cable connectors, or opening covers with lock-spring mechanisms.

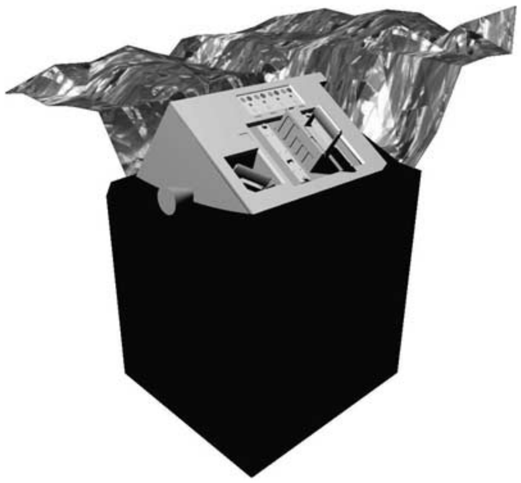
Figure 2 The virtual satellite mock-up.

3. System overview

Figure 3 shows a block diagram of the system components and the flow of information between them. Fundamental components are the physics simulation, visualization, as well as interaction interfaces, which provide the minimum set of functions of the simulation and training environment. The component's haptic control and the robot control interface provide additional features for force feedback and teleoperation, respectively. The following sections describe the components in more detail.

3.1. Physics simulation

A key element of the application is the real-time simulation of the kinematic and dynamic behaviour of the satellite components when manipulated by the user. In the current stage of the project, we assume that the servicing robot has already docked with the target satellite, so that the complex flight dynamics of the flying body dynamics can be neglected for now. We only concentrate on the simulation of rigid bodies that make up most of the virtual satellite and robot components.

In addition to the multi-body dynamics of the rigid components of the satellite, a further aspect is the simulation of the realistic behaviour of a foil for the scenario in which the MLI is removed from the satellite. This involves modelling and simulating the dynamics of bending, cutting or tearing the MLI material.

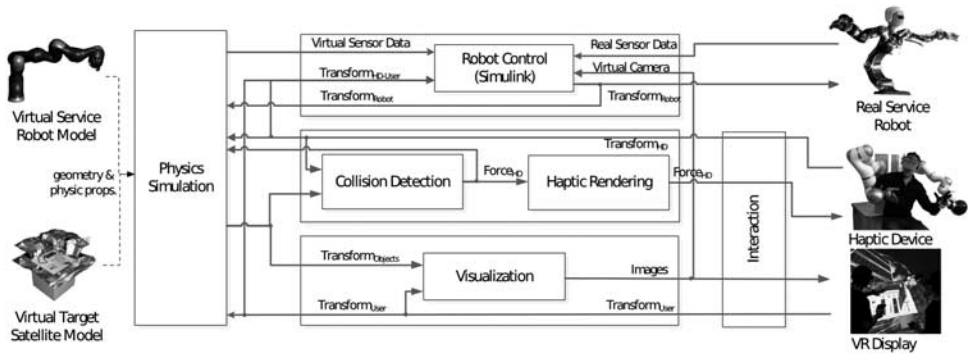

Figure 3 Block diagram of the system components and the information flow between them.

3.2. Haptic control

The haptic control component generates the necessary data for providing force feedback to the user via a haptic device. This includes the detection of collisions between the haptic interaction point (HIP) and any objects within the virtual scene, as well as the computation of the resulting force and torque affecting the HIP. The haptic device delivers the transformation of the end-effector held in the hand of the user, which is used for collision detection. In case the user grasps a virtual object, the complexity of resolving the chain of forces between the interacting objects (virtual scene, grasped object, HIP) can be reduced by attaching the grasped objects with a constant offset to the HIP and applying the resulting transformation based on the HIP's transformation. The resulting force and torque is used to update manipulated objects in the physics simulation.

The force feedback device we use is a human-scale bi-manual haptic interface (Karkee *et al*, 2011), depicted in Figure 4. It is based on light weight robot (LWR) (Hirzinger *et al*, 2002) arms mounted horizontally on a vertical column. It offers a working area similar to human arms and matches that of the humanoid robot 'JUSTIN', described in the next section. For software development and experiments, we occasionally use a Phantom Omni$^{®}$ from SensAble, which provides only translational force feedback but no torque.

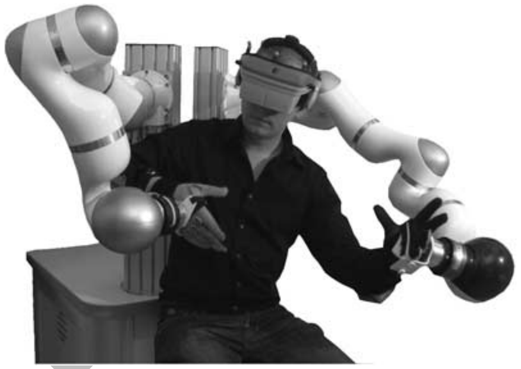

Figure 4 Bi-manual haptic human machine interface (Karkee *et al*, 2011).

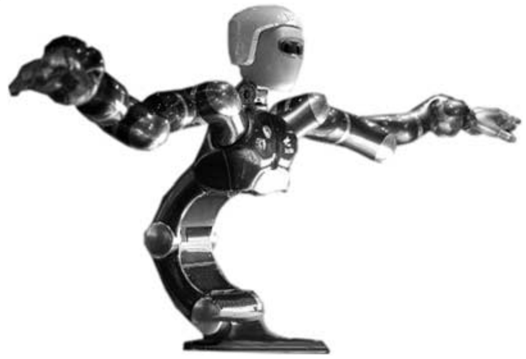

Figure 5 Humanoid service robot 'JUSTIN' (Ott *et al*, 2006).

3.3. Robot control

As mentioned above, we intend to simulate the humanoid service robot 'JUSTIN' (Ott *et al*, 2006), shown in Figure 5. It is based on two LWR arms. These are designed similar to the human arm in its size and working area, aiming at an self-weight to payload ratio of at least 1:1. A robot arm has seven revolute joints, each with motor, gear unit, power supply, force and torque sensors built-in. It reaches 0.9 m when fully stretched, weights 13.5 kg, while it can lift up to 15 kg. The robot hands are made human-like too with four joints at each finger and house 15 motors. The arms and hands are mounted to a torso, also based on LWR technology, covering a workspace similar to that of a human. The head accommodates a stereo camera, a laser-stripe sensor and an inertial measurement unit.

The control software of the robot dynamics is currently implemented in a number of Simulink$^{®}$ modules. For our simulation environment, these are accessed via an User Datagram Protocol (UDP) communication interface. The robot control reads transformations of either the haptic device or that of the tracked user, in case interaction is performed without force feedback. As output, it delivers the transformation of the robot parts in the form of a rotation angle of each robot joint, where the joints are organized in a hierarchical tree and placed at relative positions in the virtual environment. Alternatively, it can deliver the absolute location and orientation of joints when these are organized in a flat hierarchy or a list. These data are used by the physics simulation to update the rigid bodies resembling the robot parts.

Data from the robot's force and torque sensors, as well as data from the other sensors, are fed into the control unit. In the case where the robot is not connected for tele-operation but is fully simulated, these sensors have to be simulated in the form of virtual sensors. In this case, the inertial measurement unit and the laser-stripe sensor are emulated within the physics simulation, while the images of the simulated stereo camera are generated within the visualization component.

3.4. Visualization

Apart from training the correct sequence of sub-tasks, a goal of the simulation environment is to allow the user to get an awareness of the appearance and arrangement of parts and tools. Hence, the realistic and high-quality visualization of the satellite components and the environment are important factors for the success of a training simulation. This does not only include the photo-realistic rendering of detailed virtual objects with correct shading and high-resolution textures, but also the correct representation of the environmental effects that exist in orbit, such as bright sunlight, hard shadows and a moving Earth sky in the background.

Being a training and analysis tool, another research aspect of the visualization component is to augment the photo-realistic visualization of the virtual scene with the information-based, non-photo-realistic visualization of scientific data. Examples are the display of collisions between the robot and satellite parts, or the visualization of possible motion paths avoiding collisions. In addition, hints on the order of servicing sub-tasks or other instructions could be overlaid on top of satellite parts.

3.5. Interaction

In the proposed simulation and training system, the user interacts through a VR display, and optionally the haptic device. In order to support the user's feeling of presence in training, the preferred display would be an immersive display, such as an HMD, a Cave Automatic Virtual Environment (CAVE)-like surround projection-based display system. However, the system should not be bound to a specific type of display. It should scale from large multi-display VR environments to simple desktop systems. Both, the haptic device and the VR display, track the position and orientation of the user's hands for two-handed interaction. Finger tracking is provided either by using a CyberGlove® when interacting through the bi-manual haptic device, or using an optical finger tracking system. In addition, tracking the user's head is provided by the optical tracking system as part of the immersive VR display, which monitors the position and orientation of the head. This is used to render the view in the correct perspective based on the user's current viewpoint. Apart from the manipulation of virtual objects, interaction includes the navigation within the virtual environment, as well as the control and configuration of the running simulation.

Interaction within the environment should be intuitive and easy to adapt for non-expert users. In order to train realistic motion of actions, the interaction interface must be as transparent to the user as possible. If the interaction interface is too complex, for example, it could disturb and hinder the training process, or even train wrong actions. In order to aid the development of effective interaction methods, the proposed system will act as a research platform for studying novel interaction techniques and metaphors.

4. Real-time challenges

One of the main challenges of the interactive simulation environment is the modelling and simulation of the dynamic behaviour of the satellite components and their interactions with the user or the simulated robot in real-time. The computed dynamics and their kinematic effects must reflect the real conditions and behaviour of servicing a satellite in orbit. The 3D models used in the simulation environment will be taken from computer-aided design drawings of the original construction designs. These are often highly detailed and not necessarily optimized for use in a VR environment. Moreover, some mechanical mechanisms of satellite components can be quite complex, involving many individual parts. For example, the latching mechanism of a bayonet cable connector may involve threaded caps, rings, springs, several pins and holes, where all the collisions, linear and angular constraints and friction have to be resolved every simulation cycle. Computing the dynamics of several satellite components or even the complete satellite in real-time is a high target.

The requirement to visually display the results of the simulation in real-time is another challenge. The detailed complex geometric models of the satellite and robot have to be rendered in high resolution and photo-realistic quality at rates of at least 60 Hz in stereo (30 Hz per eye). Moreover, for immersive visualization, high responsiveness to viewpoint changes is crucial for the feeling of presence, and if too low can result in motion sickness (Meehan *et al*, 2003).

A further challenge is derived from the integration of haptic feedback. The haptic rendering of stiff surfaces typically requires sampling rates of at least 1 kHz to achieve sufficient stability (Mark *et al*, 1996). This puts high requirements on a fast collision detection and force computation, as the geometry of the 3D models in the scene is expected to be complex.

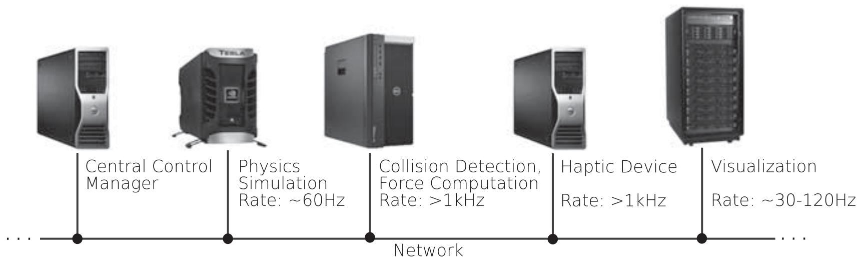
Central Control Manager
Physics Simulation Rate: ~60Hz
Collision Detection, Force Computation Rate: >1kHz
Haptic Device Rate: >1kHz
Visualization Rate: ~30-120Hz
Network

Figure 6 Distributed computation hardware setup of the simulation environment (without tele-operation component).

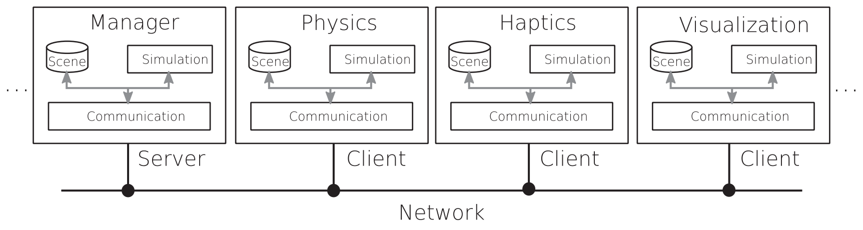

Figure 7 Logical structure of the modularized system architecture (without tele-operation module).

Finally, being an interactive simulation, any responses to actions made by the user within the virtual environment should be displayed with minimum delay. Acceptable end-to-end delay in object-focused tasks is known to be in the area of 35 ms (Ellis *et al*, 1999). Informal experiments have shown that in haptic-enabled applications, latency for calculating collision responses should be below 25–30 ms, otherwise responses may feel unrealistic (Marsh *et al*, 2006).

5. Implementation

There were two key aspects involved in designing the system's software architecture. First, the system had to fulfill the demanding computational needs for enabling the real-time simulation of a complex dynamic environment. Second, the system had to be flexible and extensible to be able to explore and experiment with different approaches for optimizations of algorithms and state-of-the-art high-performance computing hard- and software, such as supercomputers, PC-clusters, multi-core and GPGPU programming. In addition, the system had to provide an interface to easily add future subsystems. Therefore, the system was implemented in a distributed architecture, where the components of physics simulation, the haptic rendering and the visualization are transferred to separate processes running on dedicated machines interconnected through a network switch, as illustrated in Figure 6. We call such an independent component implementation, a module.

In order to aid easy development of alternative module implementations and extensions with new features, the functional structure of a module is made generic. As illustrated in in Figure 7, each module consists of a communication and a simulation process. Both are running in separate threads and exchange data via thread-safe queues in a producer-consumer pattern (Gamma *et al*, 1994). The simulation process is the heart of a module. Specific modules implement their individual functionality here. The communication process is responsible for distributing updates between the modules. The current system consists of physics simulation, haptic control, visualization and a manager module. Other modules, such as the robot control module for tele-operation and a module to enable collaboration between remote sites across the Internet within a shared simulation environment, are added in the near future. The manager module is a central component of the system that acts as server, while all other modules are clients that connect to the master and can be added and removed on demand, even at runtime.

Each module manages its own internal representation of the scene. A scene consists of a hierarchy of objects, also called nodes, each with a given state. Common state parameters include at minimum a unique identification string and a transformation matrix to describe the pose of the object within the scene. Other information, such as mass, friction or shading effects, that is specific to a particular module implementation is added to the internal node's state. For example, a physics module would internally represent the scene in a physics world with rigid bodies and constraints, while the visualization module would represent the scene with geometry nodes and shading materials in a scene graph.

All modules implement the same functional structure. Within a processing cycle, each module

1. reads state updates received from other modules;
2. interprets the messages and updates the internal scene representation;
3. steps the simulation or processes object behaviour;
4. gathers any state changes and communicates these and any other necessary status messages to the other modules.

The most important module implementations are now described in detail.

5.1. Haptics module

The haptics module is responsible for controlling the haptic device, as well as for the necessary collision detection and force computation to provide force feedback. Discrete collision detection and force computation is implemented using an extension of the Voxmap-Point-Shell^{TM} (VPS) algorithm (Renz *et al*, 2001). The algorithm utilizes two data structures, voxel map and point shell, representing the solid parts of static objects as volume-based pixels (voxels) and the surface of moving objects as a net of contact points each with a normal pointing inwards. This allows traversing the point shells efficiently to test for intersections with voxels and thus detect collisions. The penetration depth is obtained through the voxel layer. The penetration depth and the point normal yield a collision force, which is summed together to compute force and torque. With this algorithm we are able to compute collision responses within the boundary of 1 ms and meet the required 1 kHz update-rate even in highly complex scenes.

The haptic module receives the transformations of moving objects and updates the nodes in the internal representation, before starting the collision detection and force calculation in the simulation process. The transformation of the HIP and the calculated forces and torques of grasped objects are distributed to other modules by the network process.

5.2. Physics module

The simulation of rigid body physics is currently implemented using the open source real-time physics engine Bullet. Compared to other available real-time physics simulation systems, it provides good overall results (Boening and Bräunl, 2007; Hummel *et al*, 2012c). Bullet offers discrete and continuous collision detection and rigid body dynamics including various constraint solvers and generic constraints with support for constraint limits and motors. Soft body dynamics is supported via cloth, rope or deformable object structures. For accelerated internal collision detection, simple objects are approximated through basic collision shapes, such as box, sphere or cylinder, which allow for optimized collision detection algorithms. More complex objects are either decomposed into a group of several collision shapes, or their triangle mesh is used directly. In the latter case, the physics engine utilizes the efficient Gilbert-Johnson-Keerthi algorithm to perform convex collision detection. In future versions, we plan to exploit the collision detection of the VPS algorithm, as used in the haptics module, offering high performance and high accuracy of contacts with very complex geometries.

The relationship between bodies that are part of a mechanical system is defined via constraints. In Bullet, constraints are described by special joints, such as hinge, slider or generic joints with individual limits on their six degrees of freedom. These can be combined with a motor adding linear or angular force to simulate engines or springs. The mechanism of a lever switch, for example, is approximated using two rigid bodies, the base and the lever handle, connected via a hinge joint. An angular motor is applied to keep the handle to one side. If the handle is moved over its toggle threshold, the motor force is inverted so that the handle is pulled to the other side. Other mechanisms can be approximated in a similar manner.

The use of simplified rigid body dynamics and constraint management does increase simulation speed on one side. However, on the other side the simulation accuracy is compromised. Thus, a balance has to be found between simulation speed and fidelity. A solution may be a multi-rate simulation of sub-systems each with different performance settings (Karkee *et al*, 2011). The modular design of the system architecture allows us to experiment with alternative implementations of the physics simulation. In the near future, we plan to investigate the suitability of other real-time physics engines, such as PhysX^{TM} and ODE, as well as interfaces to simulation systems often found in the engineering area, such as Simulink® and Modelica.®

5.3. Visualization module

Visualization and interaction via VR displays is provided by the visualization module. Utilizing the modular system architecture, we currently developed three independent implementations of the visualization module based on InstantReality, ViSTA VR-toolkit (van Reimersdahl *et al*, 2000) and an experimental real-time ray-tracing system based on NVIDIA OptiX, each with their own specific advantages. The scene is organized in a scene graph that is continuously synchronized with state updates from the physics module and rendered during the simulation process of the module. High-quality real-time rendering is managed through the use of high resolution textures and advanced rendering techniques enabling the visualization of realistic material effects of satellite components, as well as the visualization of a realistic space environment with exact star positions and Earth with clouds and atmospheric scattering.

5.4. Manager module

The manager module hosts the central logic of the system. While the physics module handles the dynamics and kinematics of the individual parts in the simulation, the manager handles the semantics. This includes not only, for example, monitoring the on/off state of a switch, but also the management of dynamic constraints. For example, in the case of the lever switch, as mentioned above, this would be the change of the angular motor to its inverse if the handle crosses the toggle point. This semantic could of course be implemented within the physics module directly. However, as we wanted a platform to experiment with different implementations of the physics and haptic control, we wanted to remove most of the semantics from the modules in order to simplify their development.

As the physics engine is expected to implement measures for increasing stability, such as spring and dampers, the manager is responsible for the recognition and management of inter-part geometric constraints between colliding objects. It monitors the result of the physics engine for allowable rigid body motion and intervenes if violations of geometric constraints or semantic states were detected.

5.5. Communication layer

The disadvantage of using a distributed system architecture is that the necessary network communication induces delay and other negative effects due to network characteristics, such as jitter and packet loss. As mentioned above, our system adopts a client/server architecture. This means that all state updates are mediated across the manager, rather than sent directly to the target module. The reason for this was the easier management of semantics across modules. However, a consequence is that the delay between interactions and their response is double of that of a peer-to-peer architecture. Although our simulation environment (excluding tele-operation) is generally situated within a local area network (LAN), where round-trip-times of less than 1 ms are typical today, mechanisms to minimize delay should be in place. Our system provides the following features and mechanisms to minimize delay caused by communication:

Decoupled from simulation. In order to free the simulation process from message transfer, the communication is implemented in its own threaded network handler. As mentioned above, both processes exchange data via queues. Two queues are used: one for incoming messages and one for outgoing messages. The communication process uses these queues to handle messages in parallel to the simulation process. There are two communication threads per module. One thread is responsible for receiving messages from remote modules and placing them into the inbound queue. The other thread removes any pending messages from the outbound queue and distributes them. In a simulation cycle, all pending incoming messages are removed from the inbound queue and used to update the internal representation of the scene. Then the simulation is advanced one step and all changes in the scene are collected and pushed into the outgoing message queue. The queues are made thread-safe, so that no synchronization between the sending/receiving threads and the simulation is necessary.

Managed queues. The majority of messages sent in our system describe discrete updates of absolute force and/or location, for example from rigid bodies. With a continuous stream of such updates, it is preferable to communicate only the latest update, rather than to enforce that every single update is processed. Thus, the queues offer a sorting mechanism to keep only the most recent update of an object.

Vital and non-vital message types. As a most-recent sorting method cannot be applied to all types of messages, the system divides them into vital and non-vital messages. Vital messages include system commands or changes to constraint parameters, for example, while non-vital messages are continuous motion and force updates. These are then sent using reliable and unreliable protocols, respectively, via dedicated network channels.

Optional direct messages. The default method for distributing a message is to broadcast it via the manager module. In the case where no checks on semantics are necessary, a module is allowed to send a message directly to another module, without passing the manager.

Loose synchronization. In order to achieve highest performance of the local simulations in the modules, our architecture provides a loose synchronization mechanism, enabling the simulations to asynchronously run at their highest rate without needing to wait for synchronization. However, this means that update messages are generated and consumed at different rates. For example, the haptics module generates updates every millisecond, while the physics module is able to read them only every 16.67 ms, if running at the usual 60 Hz (this is often done to match the refresh rate of the visual display). If modules communicate the updates at their local processing rate, this may cause messages queuing up in the network buffers and processing queues. To overcome this problem, the system incorporates an update distribution rate control, where update messages are sent depending on the average simulation processing time of the receiving target module and the estimated network delay.

Data reduction. A common measure to reduce traffic for communicating state updates of continuous motion, such as tracking data, is to employ spatial-temporal filtering (Roberts *et al*, 2005). Spatial filtering sets a threshold on differences in translation and rotation data before issuing a new update message. However, this is not useful when subtle movements are important. With temporal filtering, an update is only sent after minimum time between distribution cycles has passed. Our system implements a combined spatial-temporal filtering mechanism on a per object basis. In order to reduce the overhead induced by the communication layer for processing single incoming and outgoing network packets, as well as to increase synchronization, several updates of the same simulation frame can be bundled into one message, as long the resulting data fits into a single packet. Further optimizations applied in tele-presence systems, such as perception-based data reduction of transmitted haptic data based on just noticeable differences (Hinterseer *et al*, 2008), may be investigated in future versions.

6. Evaluation

During implementation, the proposed system was tested on a regular basis in two VR Labs. One lab was equipped with a three-pipe active stereo 3×2 m Powerwall driven by a four-node visualization cluster, optical motion tracking and a Phantom Omni® haptic device. This setup was mainly used for developing and testing the visualization module and the underlaying framework. The other lab housed the robot and the bi-manual haptic interface and used a 42-inch passive stereo screen for the visualization. A prototype implementation of the system has also already been used in a user study, which investigated suitable interaction methods for controlling a virtual robot hand when grasping and moving the MLI foil (Hummel *et al*, 2012a, b).

In this paper, we focus on the evaluation of the real-time performance of our prototype. We measured the performance of the simulation cycles in the implemented modules, as well as the latency for transmitting messages across the distributed architecture. The following sections describe the measurements in detail, followed with the results we found.

6.1. Module performance

The performance of a module was measured in terms of simulation time and the load caused by message handling. Simulation time is the time it takes to run a full simulation cycle, including: fetching incoming updates from the inbound queue; updating and processing the simulation across all scene objects for one time step; collecting changes and pushing them into the outbound queue. In order to measure the simulation time we logged the time stamp just before triggering the simulation to advance one step and after returning from the simulation call. We also logged the elapsed time between calls to the simulation trigger to measure the time required for emptying the inbound queues and filling the outbound queues.

The number of processed messages within a simulation cycle is a measure for the load of a module. It includes the handling of received messages containing simulation updates from remote modules and handling messages with local changes to be distributed to other modules. We measured the load by logging the number of pending messages in the module's inbound queue after a receive cycle has finished, as well as logging the number of pending messages in the outbound queue before starting a distribution cycle. We provide these measurements in values of messages per second (msg/s) in the results section.

6.2. Latency

Distributed systems induce delay through the overhead of communicating state updates between the remote processes. As mentioned above, latency is an important factor in interactive object-focused tasks and can influence user experience. Hence, it is important to know how much latency our system introduces. We identified four stages of user interaction within a haptic-enabled virtual environment where latency is directly affecting user experience:

- Feeling the force when touching object surfaces.
- Seeing the virtual hand move.
- Feeling the force from moving virtual objects.
- Seeing dynamic virtual objects move.

Feeling the force when touching object surfaces occurs when moving the end-effector of the haptic device by moving the hand and immediately feeling a force feedback. This is primarily happening when touching static (non-moving) objects, but also when touching slow moving dynamic objects. In the latter case, the last known state sent from the physics module would be used for rendering haptic feedback. Delay is induced by distributing the hand position (or that of the end-effector) acquired from the haptic device to the haptics module; updating the internal state; detecting collisions and calculating the resulting force; and finally sending the force back to the haptics device.

Seeing the virtual hand move on the screen provides visual feedback of the user's hand movements. It is affected by the time it takes to send the hand position acquired from the haptics device across the haptics and the manager module to the visualization module; updating the internal scene representation; and rendering the user's virtual hand on the screen at the respective position and orientation.

The scenario of feeling the force from moving virtual objects occurs when the user feels a force feedback caused by physically simulated dynamic objects in response to user actions, such as pushing an object in the virtual environment and feeling the resistance due to the object's momentum. Here, latency is caused from passing messages of the current hand position across the system to the physics module; updating the internal scene; stepping the simulation; distributing the new positions of dynamic objects to the manager and back to the haptics module; detecting collisions and calculating the force; and sending this force to the haptics device.

Finally, seeing virtual objects move is the visual feedback from dynamic object behaviour calculated by the physics engine in response to user interaction, such as seeing a pushed object move. Similar to the scenario above, this is affected by the delay from sending hand positions across the system to the physics module; computing a simulation step; sending resulting object positions via the manager to the visualization module; and finally updating the internal scene and rendering it on the screen.

All latencies were measured in the communication layer. We did not measure the effective end-to-end latency, including delays caused by hardware for rendering visual and haptic feedback, at this point. Our system incorporates a mechanism to monitor the passing of messages between modules. It tags messages with an unique identifier allowing us to consolidate messages from logs taken on different machines. The mechanism also stores the time of creating a message. Measuring the creation time extracted from a message and subtracting it from the arrival time yields the time for travelling through the system. Furthermore, the system distinguishes messages between their types. There are three fundamental message types: pose of user (eg hand or end-effector) created by the haptic device or the visualization module; transformations of objects, either due to dynamic behaviour or in response to user interactions created by the physics module; and forces in response to collisions created by the haptics module. With these three methods, it was possible to identify messages being communicated across the system and track them from its origin to its target to determine the latency.

In 11 test runs, we measured the delay from the event of creating a movement update by the haptic device driver to receiving it in the haptics module, and the delay from creating the respective force update message until receiving it at the haptic device. We also measured the delay from creating user movement updates at the haptic device until they were received in the visualization module and in the physics module. This includes the delay induced by forwarding the message via the manager module. Furthermore, we measured the delay from creating position updates in the physics module, forwarding them via the manager, until receiving them in the haptics module, just before updating the scene and calculating the force. Finally, we measured the delay from creating resulting position updates in physics module, forwarding them via the manager, and receiving them in the visualization module, just before updating the scene and rendering.

6.3. Test environment

For the evaluation described in this paper, we set up the implemented prototype in our lab using four workstations, named *host.58*, *host.59*, *host.60* and *host.62*. The manager module was running on *host.58*, the haptics module on *host.59*, the physics module on *host.60* and the visualization on *host.62*. Each workstation had equal hardware and software components, employing two Intel Xeon Quad-Core E5530 processors, 24 GB RAM, 1 Gbit Ethernet and ran the SuSE Linux Enterprise Desktop 11 operating system. As we were mainly interested in a performance evaluation, no immersive display system nor VR interaction device was involved in the test. Instead, the visualization display used was a standard 24-inch TFT screen without stereo option. A Phantom Omni® was used as haptic device. Due to technical reasons, this was not connected directly to the machine running the collision detection in the haptics module on *host.59*. Instead, it was placed close to the visualization system, and thus connected to *host.62*. The device driver communicated with *host.59* by sending messages with the current hand position to the haptics module and receiving force messages from it via a dedicated UDP channel.

Table 1 Low-level network delay between hosts and *host.58* measured with ping

Host	*host.59*	*host.60*	*host.62*
RTT (ms)	0.209	0.186	0.229

All host machines where interconnected via a LAN using a Cisco Catalyst 4900 series switch. The workstations' clocks were synchronized using PTPd, which implements the IEEE 1588-2002 Precision Time Protocol standard. Table 1 lists the average low-level network delay as round trip time (RTT) between *host.58* and the other hosts measured with the Unix tool ping using Internet Control Message Protocol (ICMP) echo requests. Measuring the difference between timestamps written into UDP packets sent between the hosts and *host.58* confirmed that their clocks were in sync with ± 0.06 ms accuracy.

The test scene used in the evaluation consisted of a simple satellite model moving on an orbit around the Earth that had a small number of movable parts, see Figure 8. The complexity of the scene was approximately 100.000 polygons for the geometry. The scene had 11 rigid bodies: the satellite base, four module objects, one module handle, four switches each with a 1 degree of freedom rotation constraint, and a gripper representing the robot end-effector. The MLI foil was modelled using a 10×10 node flat soft body.

During the test, the spatial threshold in all moving objects was set to 5 mm for translation and 0.1 degree for rotation around all axes. With exception of the force calculation in the haptics module, the temporal threshold was set to 16.67 ms matching that of the visualization module, which was bound to the fixed update rate of 60 Hz in the renderer. Force update messages were not filtered in order to provide high update rates for haptic rendering.

7. Results

The following sections report on the results from our measurements.

7.1. Module performance

Table 2 shows the results of measuring the time it takes for executing a simulation step, as well as the time required for emptying the inbound and filling the outbound queues. The simulation processes in all modules perform at around 30–50 MHz. The processing of the inbound and outbound queues took approximately 0.02 ms in the manager and physics module. The haptics module took longer to process the queues, presumably because more messages arrived and were queued per cycle due to the high update rate of messages sent between the haptics module and haptic device. The table shows that the visualization took about 17 ms for processing the queues. As this was determined by the elapsed time between triggering the simulation, this actually includes the idle time for waiting until the next simulation call. The timing is in accordance to the fixed render refresh rate of 60 Hz.

Table 3 shows the average number of pending incoming and outgoing messages in a simulation cycle. One can see that incoming messages got queued up in the haptic module. As its simulation time was similar to that of the other modules, it is evident that significantly more messages had to be handled. For optimization, the update rate for sending messages by the haptic device could be reduced or a filtering mechanism could be applied to reduce the amount of messages sent to the haptics module. The low values for the manager and physics module indicate that there often were cycles without any pending messages. The average of one pending message in the inbound queue of the visualization module is likely to be a result of the relatively low simulation and queue processing performance, compared to the other modules. Filtering of messages sent to this module by the manager could be applied too. The value of zero for the average pending outgoing messages is due to the fact that the visualization did not create any update messages, as no tracking was used during the test.

7.2. Latency

Figure 9 shows a graph of the measured delay for sending messages from the haptic device driver to the haptics module. The delay was on average 0.625 ms (st dev = 0.244 ms). Note, that this time includes the time for messages pending in the inbound queue of the haptics module. As shown above, the haptics module has to handle a significant amount of messages, while additionally running

Table 2 Timings of executing a simulation step and required time for processing the inbound and outbound queues

Module	*Simulation step*		*Queue processing*	
	(Mean) (ms)	*(St dev) (ms)*	*(Mean) (ms)*	*(St dev) (ms)*
Haptics	0.021	0.011	0.163	0.023
Manager	0.018	0.001	0.019	0.002
Physics	0.018	0.004	0.019	0.004
Visualization	0.032	0.011	16.760	0.177

Table 3 Average number of pending messages in the inbound and outbound queues per simulation cycle

Module	*Pending inbound messages*		*Pending outbound messages*	
	(Mean) (msg/cycle)	*(St dev) (msg/cycle)*	*(Mean) (msg/cycle)*	*(St dev) (msg/cycle)*
Haptics	10.19	10.42	0.051	0.045
Manager	0.078	0.048	0.093	0.054
Physics	0.118	0.048	0.037	0.012
Visualization	1.056	0.530	0.0	0.0

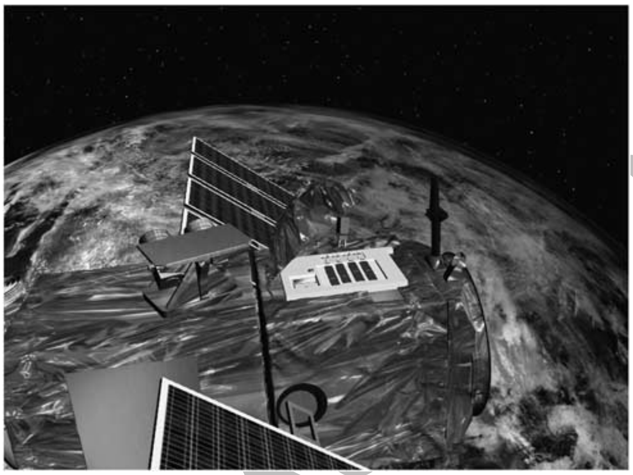

Figure 8 Snapshot of the test scene used in the evaluation.

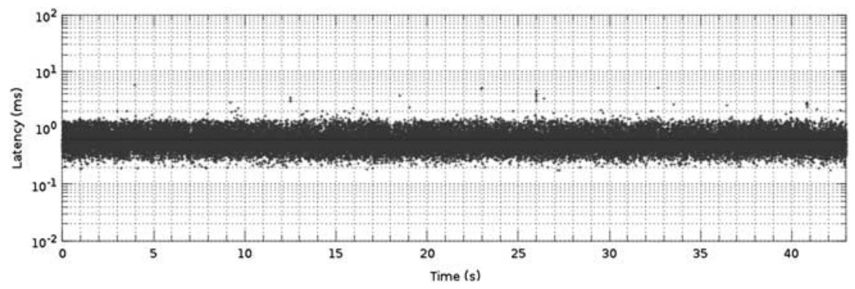

Figure 9 Delay for sending messages from the haptics device to the haptics module.

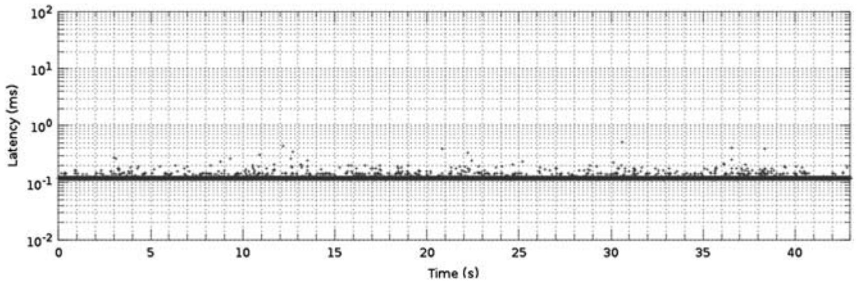

Figure 10 Delay for sending messages from the haptics module to the haptics device.

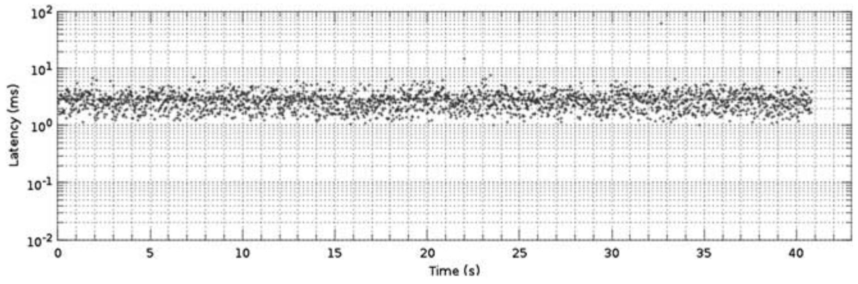

Figure 11 Delay for showing visual feedback of user moving the hand.

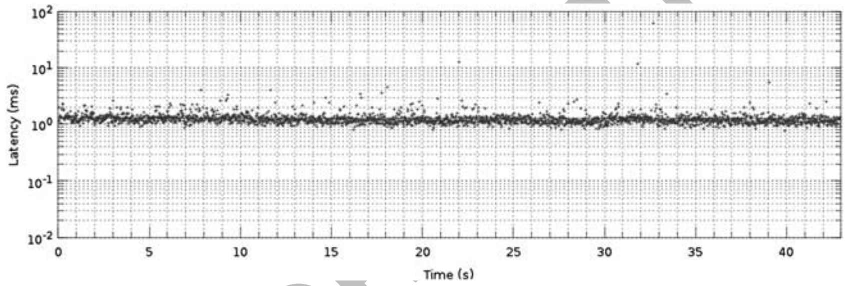

Figure 12 Delay for receiving updates of user movement at the physics module.

the simulation cycle for detecting collisions and calculating forces. The average load of handled messages per second was measured as 988.2 msg/s (st dev = 11.49 msg/s) and shows that the required 1 kHz update rate for haptic force computation can be achieved.

Figure 10 shows the delay for sending force messages from the haptics module to the haptic device. The measured latency was on average 0.119 ms (st dev = 0.004 ms). This is close to half of the RTT measured by ping in our network, as listed in Table 1, and shows that distributing the forces between force calculation and haptic device is optimal and does not introduce much additional delay. The load of handling messages was measured as 976.1 msg/s (st dev = 19.41 msg/s).

The latency for displaying visual feedback to user movement is shown in Figure 11. We measured an average latency of 2.792 ms (st dev = 1.604 ms) with an upper bound of 8 ms, and a lower bound of 1 ms. The average load of handled messages was 57.5 msg/s (st dev = 0.88 msg/s), which confirms the fixed update rate of the visualization process.

In the graph in Figure 12, we show the latency of user movement messages arriving at the physics module. With an average delay of 1.281 ms (st dev = 1.283 ms) it is double of that of acquiring these messages from the haptic device to the haptics module, as it includes forwarding through the manager module. However, the message load was measured as 57.6 msg/s (st dev = 0.70 msg/s). This shows that the initially high load of incoming data from the haptic device was successfully reduced to meet the target simulation rate in the physics module. It is the effect of the temporal filtering mechanism applied to reduce the overall message traffic in our system.

The latency for transferring new object positions from the physics module to the haptics module in response to user actions, shown in Figure 13, was measured as 1.219 ms (st dev = 0.291 ms) delay on average and an average load of 54.0 msg/s (st dev = 2.55 msg/s). On some occasions, the latency doubles or triples. This seems to be due to increased

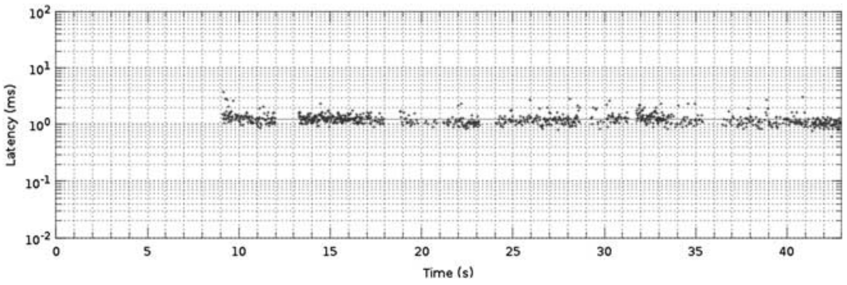

Figure 13 Delay for transferring new object positions from the physics module to the haptics module.

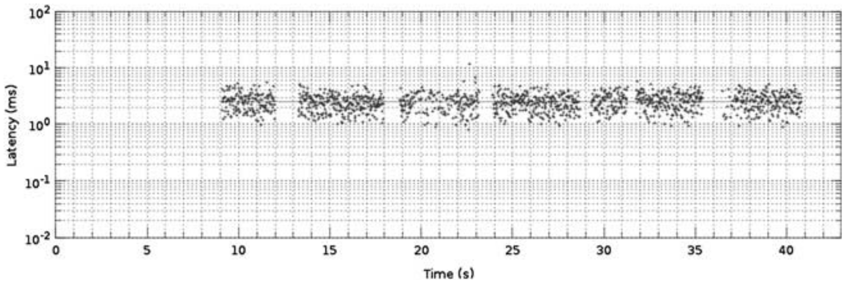

Figure 14 Delay for transferring new object positions from the physics module to the visualization module.

Table 4 Estimated end-to-end latency in the four stages of haptic interaction

Stage of user interaction	*Estimated latency (ms)*
Feeling the force when touching object surfaces:	0.8
Seeing the virtual hand move:	6.0
Feeling the force from moving virtual objects:	2.8
Seeing dynamic virtual objects move:	7.1

load from messages sent by the physics when the object collides with other objects, which create more movement updates.

Finally, Figure 14 shows the latency for transferring new object positions from the physics module to the visualization module. Similar as in Figure 11, this graph shows the effect of pending messages waiting in the inbound queue in between render cycles. The average latency was 2.553 ms (st dev = 0.931 ms) with an upper bound of 5.3 ms and a lower bound of 0.9 ms. The load of handling messages was 58.7 msg/s (st dev = 1.17 msg/s).

On the basis of our measurements, we can make estimations on the communication-level end-to-end latency induced by our implemented system in the four identified stages, as shown in Table 4.

8. Conclusion

This paper introduced a real-time interactive simulation and training environment used as a platform for the analysis, training and programming of OOS tasks. The aim is to allow performing manual and robotic assembly and disassembly tasks within a haptic-enabled immersive virtual environment. The challenges are the real-time simulation and rendering of the correct dynamic behaviour and the realistic appearance of satellite components and their complex mechanisms resembling the real conditions in space, as well as to integrate interaction through haptic devices. The paper described our proposed distributed system architecture utilizing dedicated computing resources to fulfill the real-time constraints. The architecture divides the physics simulation, visualization and haptic rendering into separate modules that run in parallel on dedicated machines. A central manager mediates the communication of state updates, while managing the global semantics of object behaviours.

In a system performance evaluation, we measured the timings of simulation cycles, as well as the processing load based on the number of messages handled per second. We further measured the delay caused for communicating data between simulations across the distributed system. It was shown that our system is capable of providing the required high update rates of 1 kHz for haptic rendering and 60 Hz for visual rendering. On the basis of measurements of latency at several communication paths, we find an end-to-end latency for presenting visual and haptic feedback for user movement and movement of simulated dynamic virtual objects in response to user interaction of on average 7.1 ms and 2.8 ms, respectively.

The modular design of the proposed architecture facilitates the research and development of improvements and extensions with focus on individual aspects of the simulation environment. Future work will investigate alternative real-time physics engines, develop optimizations to the haptic rendering and explore two-handed interaction techniques.

References

Artigas J, Kremer P, Preusche C and Hirzinger G (2006). Testbed for telepresent on-orbit satellite servicing. In *Proceedings of the Human-centered Robotic Systems Conference (HCRS)*; Munich, Germany.

Boening A and Bräunl T (2007). Evaluation of real-time physics simulation systems. In *Proceedings of the 5th international conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia (GRAPHITE)*; Perth, Australia, pp 281–288.

Currie NJ and Peacock B (2002). International space station robotic systems operations—A human factors perspective. In Proceedings of the *Human Factors and Ergonomics Society* **46**(1): 26–30.

Ellery A, Kreisel J and Sommer B (2008). The case for robotic on-orbit servicing of spacecraft: Spacecraft reliability is a myth. *Acta Astronautica* **63**(5–6): 632–648.

Ellis SR, Young MJ, Adelstein BD and Ehrlich SM (1999). Discrimination of changes of latency during voluntary hand movement of virtual objects. In *Proceedings of the Human Factors and Ergonomics Society*. **43**(22): 1182–1186.

Gamma E, Helm R, Johnson R and Vlissides JM (1994). *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Professional: Boston, MA.

Hinterseer P, Hirche S, Chaudhuri S, Steinbach E and Buss M (2008). Perception-based data reduction and transmission of haptic data in telepresence and teleaction systems. *IEEE Transactions on Signal Processing* **56**(2): 588–597.

Hirzinger G, Brunner B, Dietrich K and Heindl J (1994). ROTEX–The first remotely controlled robot in space. In *Proceedings of IEEE Internationl Conference on Robotics and Automation (ICRA'94)*; San Diego, CA.

Hirzinger G, Sporer N, Albu-Schäffer A, Krenn R, Pascucci A and Schedl M (2002). DLR' torque-controlled light weight robot III—Are we reaching the technological limits now? In *Proceedings of IEEE Internationl Conference on Robotics and Automation (ICRA2002)*; Washington DC, pp 1710–1716.

Hulin T, Sagardia M, Artigas J, Schaetzle S, Kremer P and Preusche C (2008). Human-scale bimanual haptic interface. In *Proceedings of the 5th International Conference on Enactive Interface*; Pisa, Italy.

Hummel J, Wolff R, Gerndt A and Kuhlen T (2012a). Comparing three interaction methods for manipulating thin deformable virtual objects. In *Proceedings of IEEE Virtual Reality*; Costa Mesa, CA, pp 139–140.

Hummel J, Wolff R, Dodiya J, Gerndt A and Kuhlen T (2012b). Towards interacting with force-sensitive thin deformable virtual objects. In *Proceedings of EGVE – EuroVR Joint Virtual Reality Conference of ICAT (JVRC'12)*; Madrid, Spain, pp 17–20.

Hummel J, Wolff R, Stein T, Gerndt A and Kuhlen T (2012c). An evaluation of open source physics engines for use in virtual reality assembly simulations. In *Lecture Notes in Computer Science* Vol. 7432. Springer: Berlin Heidelberg, pp 346–357.

Karkee M, Steward BL, Kelkar AG and Kemp II ZT (2011). Modeling and real-time simulation architectures for virtual prototyping of off-road vehicles. *Virtual Reality* **15**(1): 83–96.

Landzettel K *et al* (2006). ROKVISS verification of advanced light weight robotic joints and tele-presence concepts for future space missions. In *Proceedings of the 9th ESA Workshop on Advanced Space Technologies for Robotics and Automation (ASTRA)*, ESTEC; Noordwijk, Netherlands.

Mark WR, Randolph SC, Finch M, van Verth JM and Taylor II R (1996). Adding force feedback to graphics systems: Issues and solutions. In *Proceedings of ACM SIGGRAPH Computer Graphics and Interactive Techniques*; New Orleans, Louisiana, USA, pp 447–452.

Marsh J, Glencross M, Pettifer S and Hubbold R (2006). A network architecture supporting consistent rich behavior in collaborative interactive applications. *IEEE Transactions on Visualization and Computer Graphics* **12**(3): 405–416.

Meehan M, Razzaque S, Whitton MC and Brooks Jr FP (2003). Effect of latency on presence in stressful virtual environments. In *Proceedings of IEEE Virtual Reality*, Los Angeles, CA, pp 141–148.

Ott C *et al* (2006). A humanoid two-arm system for dexterous manipulation. In *Proceedings of IEEE-RAS International Conference on Humanoid Robots*; Genova, Italy, pp 276–283.

Renz M, Preusche C, Potke M, Kriegel H and Hirzinger G (2001). Stable haptic interaction with virtual environments using an adapted voxmap-pointshell algorithm. In *Proceedings of the Eurohaptics*, Birmingham, UK, pp 149-154.

Roberts D, Marshall D, McLoone S, Delaney D and Aspin R (2005). Exploring the use of local inconsistency measures as thresholds for dead reckoning update packet generation. In *Proceedings of IEEE Distributed Simulation and Real-Time Applications*; Montreal, Canada, pp 95–102.

Rupp T, Boge T, Kiehling R and Sellmaier F (2009). Flight dynamics challenges of the german on-orbit servicing mission DEOS. In *Proceedings of the 21st International Symposium on Space Flight Dynamics*; Toulouse, France.

Sellmaier F, Boge T, Spurmann J, Gully S, Rupp T and Huber F (2010). On-orbit servicing missions: Challenges and solutions for spacecraft operations. In *SpaceOps 2010 Conference*, American Institute of Aeronautics and Astronautics; Huntsville, Alabama, USA.

van Reimersdahl T, Kuhlen T, Gerndt A, Henrichs J and Bischof C (2000). Vista: A multimodal, platform-independent vr-toolkit based on wtk, vtk, and mpi. In *Proceedings of the 4th International Immersive Projection Technology Workshop (IPT)*; Ames, Iowa.

Received 23 April 2012;
accepted 28 March 2013 after one revision