XPERSim: A simulator for robot learning by experimentation*

Iman Soliman Awaad[1] and Beatriz León[2]

[1] Bonn-Rhein-Sieg University of Applied Sciences
Grantham-Allee 20, 53757 Sankt Augustin, Germany
iman.awaad@fh-bonn-rhein-sieg.de
[2] Universitat Jaume I,
Castellon de la Plana, Spain

Abstract. In this paper, we present XPERSim, a 3D simulator built on top of open source components that quickly and easily constructs an accurate and photo-realistic simulation, both visually and dynamically, for robots of arbitrary morphology and of the environment within which it functions. While many existing robot simulators provide a good dynamics simulation, they often lack the high quality visualization that is now possible with general-purpose hardware. XPERSim achieves such high quality visualization by using the Object-Oriented Graphics Rendering Engine 3D (Ogre) engine to render the simulation whose dynamics are calculated using the Open Dynamics Engine (ODE). Through XPERSim's integration into a component-based software integration framework, XPERSIF, and the use of the scene-oriented nature of the Ogre engine, the simulation is distributed to numerous users, thus enabling simultaneous, quasi-realtime observation of the multiple-camera simulations.

1 Introduction

Robot simulators are widely used in the robotics field for different purposes. They have mainly been used to design and test new robot models as well as to develop the necessary software for running the robots, such as controllers or behaviors. The simulation of multi-robot teams [1], for example, is a vital tool in fields such as Robocup, where the setting up of a whole team of robots is a time-consuming task. Likewise, the simulation can be run for as long as is needed and is not limited by physical constraints such as battery life. In this way, simulation also contributes to speeding up the pace of research. Where multi-robot teams are concerned, a simulator that allows the testing of team behaviours is ideal. A 2D

* The work described in this article has been partially funded by the European Commission's Sixth Framework Programme under contract no. 029427 as part of the Specific Targeted Research Project XPERO ("Robotic Learning by Experimentation"). The authors express their gratitude to Keyan Ghazi-Zahedi, Ronny Hartanto, Karl-Heinz Sylla and Paul Ploeger for their guidance and to the researchers in the XPERO project for their feedback and support.

simulator is often sufficient for this purpose. The field of evolutionary robotics also relies heavily on simulation, as the time spans used for their purposes are generally very long. In this special case, a fast simulation is the highest priority.

The quality of a simulation is largely dependent on the physics engine which calculates the dynamics of the simulation, and the rendering engine which is used to visualize the simulation. The results of the physics simulation are highly dependent on the accuracy of the models which are provided by the user. There are many physics engines available with varying quality and cost. Similarly, a wide variety of 3D rendering engines, used for visualization, also exist. The game industry has helped to advance the quality of these engines to its current limits; to the point where open-source engines that provide this exceptionally high-quality visualization are now available.

In the above-mentioned cases, the simulation is used by the researcher to visualize the behavior of the robots and not by the robot itself (as is the case in the XPERO project, which deals with robot learning by experimentation and for which XPERSim was developed). Within the project, the simulation is used by both the researchers and the robot itself. For the robot to function as expected, its perception of its environment should be as realistic as possible, both dynamically and visually. The dynamics of its environment must use accurate models of friction, mass, forces, rigid body collisions, and so on. In addition, the dynamics have to allow for an accurate simulation of the manipulation process itself. Realistic visualization of this interaction with its environment is vital for the observation and the perception processes. The robots use a variety of vision techniques and mechanisms such as focus of attention and novelty detection which allow them to autonomously find objects to experiment with. In order for these techniques to be tested and used in a simulated environment, it is necessary that the visualization be as realistic as possible. The use of lighting, shadows and textures contribute to this realism.

XPERSim provides a realistic and accurate physics simulation that is also visually realistic at a reasonable computational cost. It achieves high quality visualization by using the Ogre 3D engine to render the simulation whose dynamics are calculated using ODE. This enables an accurate simulation of robot observation and manipulation tasks. It also enables the replication of experiments, regardless of whether the robot is physically present or not, and collaboration across geographical borders. In this paper we describe our simulation of the the Khepera [2] robot from K-Team and the XPERO environment in which it functions as created using XPERSim. We will first give a brief overview of 3D robot simulators. We will introduce the architecture of XPERSim and introduce the contents of the packages. We will then discuss the advantages of using the Ogre 3D rendering engine and the ODE physics library as well as the challenges and results of integrating these technologies before presenting the methods used to distribute the simulation to multiple users simultaneously in a quasi-realtime manner. The results are then presented. Finally, we conclude with a discussion of the work.

2 Related work

There are numerous 3D robot simulators available such as Gazebo [3], USARSim [4] and Webots [5]. Many of these robot simulators use ODE for their dynamics simulation. ODE is a free, open-source, high-performance library for simulating rigid body dynamics. It is stable, mature and platform-independent with an easy to use C/C++ application programming interface (API). It has advanced joint types and integrated collision detection with friction [6]. ODE's major drawback is that of the quality of rendering done through the DRAWSTUFF library that comes with it. It should be noted that the DRAWSTUFF library is provided by the authors of ODE for debugging purposes and is not meant to be used for a simulation. In this section we briefly survey a number of 3D robot simulators.

Gazebo is part of the Player/Stage project, one of the leading tools in the robotics field. It comes with a large library of sensors and models of existing robots. These can be controlled by either the Player server or by controllers provided by the user. To create ones own robot requires code-based modeling of the robot in C [3]. Gazebo's dynamics simulation is based on the ODE library.

Webots from Cyberbotics Ltd. is a commercial simulator capable of simulating many kinds of mobile robots. Features include a complete library of sensors and actuators, the ability to program the robots in C, C++, Java or third party software and the use of the ODE library for physics simulation. It also comes with models of some commercially available robots. In addition, there is a robot and world editor that enables the user to create the environment and the robot from the libraries mentioned above [5]. However, the visualization is not of a sufficiently high quality.

USARSim is a high fidelity simulation of urban search and rescue (USAR) robots and environments intended as a research tool for the study of human-robot interaction (HRI) and multirobot coordination [4]. It uses the Unreal game engine for the dynamics simulation and the visualization. The physics engine used by Unreal is the Karma engine. The visualization however, is of far superior quality than that of the above mentioned simulators.

3 Appraoch

To solve the problem of ODE's limited-quality visualization, the Ogre 3D engine was chosen to perform the rendering. Ogre 3D is a free, open source 3D engine written in C++, which is designed to make it easier and more intuitive for developers to produce applications using hardware-accelerated 3D graphics. It is important to note that Ogre 3D is not a complete simulation engine. It performs many tasks, but most of them are related to 3D computer graphics. It does not, for example, provide physics, sound, networking, GUIs or artificial intelligence (AI). There are, however, other libraries from which one can choose to perform these tasks [7]. The core concept of Ogre is the "scene". Within this scene, the "root" object is the entry point to the Ogre system [7]. It maintains pointers to all objects in the system, such as scene managers and resource managers.

These give access to individual entities within a scene. Each entity is attached to a "scene node". The root object also contains a method that is in charge of looping to render continuously. A scene-graph (a collection of nodes in a graph or tree structure) is created at the beginning of a simulation and is maintained throughout. Each frame, this graph is traversed and its entities rendered, thus producing the simulation. With each iteration of the simulation loop, the position as well as the orientation of the entities to be simulated is retrieved from ODE and rendered using Ogre.

The conceptual model of XPERSim is simple. It has a client-server architecture where the client controls the robot in the simulation which is running on the server-side. The client can be a console running on Windows or any other platform. The server can be interfaced with a client in the form of an AI program or a planner which would then control the robot. The current version of XPERSim implements the Khepera robot from K-Team Switzerland.

3.1 Implementation

XPERSim provides a library of model components, written in C++, that are useful for robot simulation. Its modular architecture also allows for maximum code reuse and makes it open for expansion.

Any simulation requires a robot and the environment on which it will act. Setting up the environment requires very basic knowledge of ODE. A wrapper encapsulates the ODE function calls necessary for the creation of the entities as well as the storing away of the information that will be used by Ogre to render it. It acts as a simple interface between the user and the wrapped code. The information needed to render each object with a specific mesh and parenthood is stored and retrieved later on to create the scene-graph that Ogre will use to render these bodies. The contents of the simulation are categorized as being either part of the environment or part of the robot. Objects which may be rendered are placed within description arrays and are updated each frame. Joints, while critical to the physics of the environment (in this case the joints are more likely to be used to simply hold together walls and will never have forces applied to them to actuate them), it is not desirable to have them rendered. As such, they are not considered entities and are not saved within the description array. The robot differs from the environment in that it contains actuators and sensors in addition to rigid bodies and non-actuated joints. These need their own descriptions to facilitate retrieving sensor values and sending commands to the actuators. Two packages have been created specifically for sensors and actuators. In addition, the method of communication with each embodiment will differ. For this reason, each Robot contains its own communicate function.

3.2 Actuators and Sensors

The Actuator package contains a number of actuators, namely a differential drive, the Khepera arm and the gripper have been implemented. The Sensor package currently implements a number of sensors, such as the IR proximity

sensor, light barrier sensor, touch sensor, wheel encoder and the camera. The light barrier and touch sensors are simulated using IR sensors.

The IR sensors have been implemented using five rays, all with the same start position. This implementation was provided by [8]. One ray lies in the exact orientation given. Two of the remaining rays are directed at orientations that take into consideration a spread angle on the x-axis, while the remaining two rays take into consideration the spread angle in the y-axis. In this way, a cone is created that more accurately emulates an IR sensor's field. The spread angles are parameters that the user is able to set, as is the length of the ray which is set to the sensitivity of the IR sensor being simulated. This is an advantage over other simulators, such as Webots, which uses only one ray for a an IR sensor. This method of modeling the sensor with five rays also allows a more realistic sensor model to be created. The real sensor detects a distant object, if a close object penetrates the cone less than halfway. If one of the other rays is activated, a weighted sum could be used to calculate the distance instead of the minimum value [8]. By varying the spread-angles of the rays, the sensor model can be changed to reflect a real sensor whose values have been obtained, or to simulate noise. By gradually changing the parameters, a transition can be made from the idealistic simulated world to the real world.

The XPERSim window contains two viewports, each displaying the view from a specific camera. The "overhead' camera is displayed on the right while the left half of the window displays the "first-person" camera attached to the robot. Ogre allows the user to add as many viewports as is needed and as many cameras. This feature can be used to easily simulate stereo-vision. While the rendering is done in all the viewports, XPERSim currently allows the user to move the "overhead' camera only. It is possible to save and retrieve rendered frames to and from a file. This means that it is also possible to apply vision algorithms to these frames, or transmit them over a network to users. A perception module with basic vision algorithms was implemented for the Logging version of XPERSim which allowed the frames to be analyzed in an off-line manner. The process of saving a file to disk is however a costly process as the image must first be flushed from the GPU.

The communication framework enables the simulation running on the server-side to communicate with a client-side console over TCP/IP. It is robot-specific. For this reason, the communication for the Khepera robot is included within the class implementing it.

4 Distributing the simulation

This section details the efforts made to distribute the simulated images for tele-observation. Although the implementation is specific to the XPERSim simulator, the same approach could conceptually be used for other simulators. XPERSim has been integrated into the XPERSIF framework, a component-based software integration framework which was specified, defined, developed, implemented and tested by the authors. The framework and architecture comprise loosely-coupled, autonomous components that offer services through their well-defined interfaces

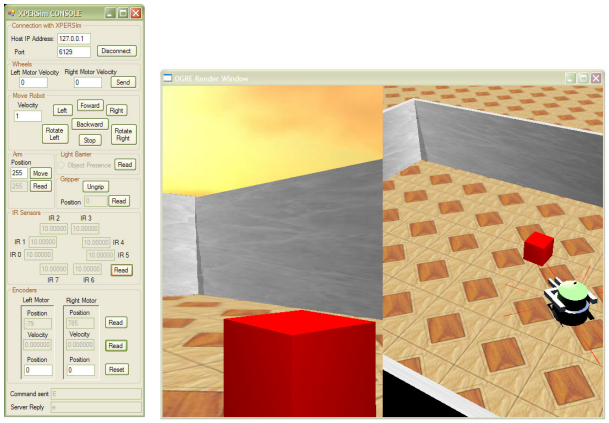

Fig. 1. A screenshot of XPERSim running alongside the console. On the left-hand side is the console to control the simulated Khepera and read sensor values. On the right is the XPERSim window, with the two viewports. The one on the left shows the view from the Khepera's perspective, while the one on the right displays the experimenter's view. The user is able to use the mouse and keyboard to move this camera.

and form a service-oriented architecture. The Ice middleware is used in the communication layer. This integration of XPERSim into XPERSIF allows it to be run in a distributed setting.

While tele-operation and tele-observation of the simulation were previously implemented, the solution for tele-observation was provided with a focus on fulfilling a use case for data generation which provided traces for the machine learning tools. These traces included the simulated image which was requested and transmitted through a synchronous RPC call. While this requirement was met, the solution did not enable a frame-by-frame view of the simulation. The specification of new use cases specified the need for the architecture to supply quasi-real time observation of the simulated image. The implementation of the solution is presented in this section. A number of issues precluded the use of the same method for true real time tele-observation of the experiment:

- The presence of a bottleneck in obtaining the rendered image from the GPU (Graphics Processing Unit) to the CPU which makes the process of simply obtaining the image a time-consuming affair.
- The transmission of the image itself takes time.

It should be noted that these issues made infeasible the real time or quasi-real time tele-observation of the experiment by even one single client. In order to facilitate scalability, bottlenecks must be avoided.

The solution presented here, which bypasses this bottleneck, uses a proven methodology (often implemented in multi-player games) which involves moving the rendering of images from the server-side to the client-side by sending out a subset of the scene information to ensure that all clients are operating syn-

chronously [9], thus drastically reducing the amount of data being transmitted. This is possible due to the scene-oriented nature of the XPERSim simulation. As mentioned previously, the Ogre 3D rendering engine uses scene-graphs to represent hierarchies, which simplifies the processing of objects or groups of objects. A scene-graph consists of nodes (with parent nodes and child nodes). If a parent node is translated or rotated, this transformation is applied to the child scene nodes as well. The latency resulting from the distributed nature of the application is ameliorated by sending the node information from the simulator while the client is rendering the previous one – i.e. the server does not wait for the client to request the image but sends it continuously once it has subscribed. The method described above to distribute a simulation to multiple clients is implemented here by decoupling the physics and graphics engines from XPERSim to create an XPERSim Server (calculating dynamics) and a TeleSimView client (rendering the nodes at their new positions). The XPERSim Server sends out the positions and orientations of all scene-nodes to the clients that simply transform the specified nodes to the specified positions and orientations and in so doing produce the same scene in an efficient and real time manner. In the refactored implementation of the XPERSim simulator, no distinction is being made between parent nodes and child nodes. It is recommended however that this distinction be made as it would reduce the number of nodes whose data needs to be transmitted (transmit parent nodes only and nodes which can be moved separately from the hierarchy – a gripper for example which, despite being a member of the robot node, may be moved on its own). The details of this implementation are described below.

4.1 XPERSim Server

As mentioned above, the XPERSim Server is now solely responsible for calculating the dynamics of the simulation and for their distribution to the clients. The separation of the two engines was straightforward due to the modular structure of the simulator. The first step was the removal of a Windows-specific thread and mutex implementation and its replacement with Ice threads and mutexes. Previously, every rendered frame would step the simulation by 0.05 seconds (5 x 0.01 seconds). With this link to the rendering of a frame gone, the speed at which the simulation proceeded much faster. Various methods for transmitting the node information were evaluated.

During the start-up of the simulation and the creation of the ODE bodies, the information pertaining to the Ogre-scene is accumulated. This information is stored in a container structure which is requested by the Camera subcomponents which will publish the images. The same scene is rendered from each camera's position. As the robot's camera is attached to it, it will automatically be moved when the robot moves. If a pan/tilt camera is used, then its position and orientation could be sent out as a node.

In an effort to further reduce network latency, a one-way invocation is used to send the new frame. This can in fact be quite expensive when many such small messages need to be sent. This is because the run time taps into the OS kernel

for each message and because each of these messages is sent out with its own message header [10]. To ameliorate this problem, batched one-way invocations are used. This allows the Ice run time to buffer these small messages until the XPERSim Server explicitly flushes them.

Originally, it was envisioned that the parametrization of XPERSim would be done through an XML file. This would allow the client to send the setup for a new experiment without necessitating the recompilation of XPERSim. The limited number of scenarios and the low frequency at which these scenarios are changed dispenses with the need for the XML parametrization and makes it equally efficient to choose precompiled setups.

4.2 TeleSimView Client

The TeleSimView client is used to view the simulated scene. With the same node information, the view from both cameras is rendered. The subscription to receive the node information is made with the XPERSIF components: PERCEPTION (robot camera) or OBSERVATION (overhead camera).

The TeleSimView client only requires Ogre (and its dependencies). Ogre itself has always been cross-platform compatible. The source code for a project running under Windows could not previously be compiled and used directly on other platforms however due to the use of Windows-specific libraries handling events and key input. With the release of Ogre version 1.4.6 (a.k.a. 'Eihort'), this problem is now solved with the use of the Object-oriented Input System (OIS) platform.

A two-way invocation to the PERCEPTION (or OBSERVATION) component fetches the scene which will be created and subsequently updated. The creation of the scene involves the creation and attaching of nodes, their positioning and the creation of such basic scene items as the plane, lights, and sky. Once this has been done, the client uses operations found within the interfaces which are extended by the PERCEPTION and OBSERVATION components in order to subscribe as an image-observer. As soon as this is done, the images will be transmitted to it from the relevant camera subcomponent (the image-provider).

5 Results

The XPERO project has provided XPERSim with an invaluable testing ground. XPERSim has proven to be highly useful and effective in speeding up the pace of research. This has been made even more evident within the distributed research environment. XPERSim has been successfully used to aid the human researcher in developing and evaluating concepts as well as providing test data [11] by using the initial Logging version of XPERSim, in addition to subsequent versions following its integration into XPERSIF. A perception module has been developed as proof of concept that allows basic vision algorithms to be performed on the simulated scene. The Client Console (implemented with a TCP/IP connection) enables tele-operation of the simulated robot using the same commands that are

sent to the physical robot. In this way, any user with code to control a Khepera can use this code in XPERSim. Simultaneous multiple camera simulation of the rendered scene is possible at high frame rates. A library of components that can be parametrized by the user has been created. This library includes a number of commonly-used sensors and actuators.

Due to the modular architecture of the simulator, it should be possible to easily simulate multiple robots by making minor additions to the structure of XPERSim. The number of simultaneous camera simulations is limited by the maximum resolution of the screen if real-sized viewports are required for the "first-person" cameras. The frame-rate is mainly affected by the number of objects within a scene and the number of triangles used in the mesh used to visualize it. A slowdown in the frame-rate usually occurs when many hundreds of nodes are being visualized [12]. There are many optimizations that can be made within Ogre to help in situations where these numbers are very large. Many are available to download from the Ogre website. A potential bottleneck exists in flushing the buffer in the graphics card to save the rendered image. If this is done often, for example for logging purposes, the simulation speed slows down.

Distribution of the simulation through its integration into the XPERSIF architecture was successfully achieved. The scalability of the implementation described above was evaluated by measuring the impact on the quality of the simulation by varying the number of subscribers to the tele-observation service. This detailed scientific evaluation validated the use of a batched one-way invocation for distributing the image. Table 1 shows the measurements made when one, three, five and then ten clients are subscribed to the service. All experiments were repeated three times, measuring the time it took for 60 frames to be delivered to the TeleSimView client. It is worth noting that the size of the image to be rendered is inconsequential. As nodes are being sent and not an image, it is the number of nodes within a scene that impacts the time and not the image size. For the test case above, 15 nodes were transmitted (representing the Khepera robot and the four cubes. Using this information, the scene may be rendered from the viewpoint of any number of cameras.

Trial	1 client	3 clients	5 clients	10 clients
1	0.0039 s	0.0039 s	0.0219 s	0.0227 s
2	0.0023 s	0.0172 s	0.0128 s	0.0352 s
3	0.0075 s	0.0036 s	0.0120 s	0.0448 s
Mean	0.0046 s	0.0082 s	0.0156 s	0.0342 s

Table 1. The time in seconds between receiving two subsequent images using batched one-way invocation methods.

6 Discussion

XPERSim enables the replication of experiments, regardless of whether the robot is physically present or not. It has been used successfully, not only in the initial stages of the project in allowing the researchers to pursue multiple scenarios simultaneously to develop and evaluate concepts, but also in the later stages by providing vital traces used for the machine learning process. The initial results [11] from the XPERO project support the original assertion that simulation has indeed enhanced the speed of research within the project. XPERSim has the advantages of providing a more realistic camera simulation at faster than real-time frame rates and a library of available model components that are useful for robot simulation and include realistic sensor models. It is modular, extensible, easy to use and understand and provides logging functionality. It enables distributed work without the need for a physical robot and enables easy replicability. We have addressed the problem of tele-observation by decoupling the physics and rendering components within the simulator in a manner that optimizes computational power and harnesses the power of node-oriented scene-graphs, and thus reduced network latency. The extension of the library to include more robot models, sensors and actuators is a top priority on our agenda. By using the Ogre 3D engine and ODE as base components in our simulator's architecture, we have produced a simulation with accurate physics and high quality graphics that can be used with great ease and without the use of special hardware.

References

1. Ziparo, V.A., Kleiner, A., Nebel, B., Nardi, D.: RFID-based exploration for large robot teams. In: Proc. of the IEEE International Conference on Robotics and Automation (ICRA). (2007)
2. KTeam: Robot Khepera II Specifications. (March 2006)
3. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. (March 16 2004)
4. Wang, J.: USARSim: A Game-based Simulation of the NIST Reference Arenas. (2006)
5. KTeam: Webots 3D Robotic Simulation. (March 2006)
6. Smith, R.: Open Dynamics Engine. (2006)
7. Team, T.O.: OGRE Manual v1.2.0 ('Dagon'). (2006)
8. Ghazi-Zahedi, K.: Self-regulating neurons: A real-time learning algorithm for recurrent neural networks. PhD thesis, University of Osnabrueck (2008 (to appear))
9. Forums, O.: Streaming a scene rendered by a camera. Online at http://www.ogre3d.org/phpBB2/viewtopic.php?p=224646 (May 2007)
10. Henning, M., Spruiell, M.: Distributed Programming with Ice. ZeroC Inc. Revision 3.2 edn. (March 2007)
11. Bratko, I.: Initial experiments in robot discovery in xpero. In: ICRA 2007 Workshop on "Concept Learning for Embodied Agents". (April 2007)
12. Forums, O. Online at http://www.ogre3d.org/index.php?option=com_content&task=view&id=196&Itemid=97