LiveInventor: An interactive development environment for robot autonomy

Charles Neveu, QSS Group, Inc
neveu@artemis.arc.nasa.gov
Mark Shirley, NASA Ames Research Center
shirley@ptolemy.arc.nasa.gov
MS 269-3
NASA Ames Research Center
Moffett Field, CA 94035-1000 USA

Keywords robotics, autonomy, physically-based simulation

Address correspondence to:
Charles Neveu, MS-269 NASA Ames Research Center, Moffett Field, CA, 94035-1000 USA
Phone: 650-604-2525 FAX: 650-604-4036

Introduction

LiveInventor is an interactive development environment for robot autonomy developed at NASA Ames Research Center. It extends the industry-standard OpenInventor graphics library and scenegraph file format to include kinetic and kinematic information, a physics-simulation library, an embedded Scheme interpreter, and a distributed communication system.

Background and motivation

Reliable, robust autonomy is crucial for long distance and long duration unmanned exploration of the Martian surface, but autonomy is difficult to put on mission for a number of practical reasons. Like any engineers, autonomy developers require a platform on which to test and debug their product. For Mars missions this generally means one of two unique, identical rovers: one that flies and one that stays on the ground, shared by all hardware and software teams for test and simulation. Autonomy development and testing generally needs a complete, operational rover and must wait for all other teams to complete their work before really being able to begin theirs. Operating the rover in a sandbox for autonomy development and testing is an expensive, labor-intensive and time-consuming proposition. Even then, a duplicate rover in a sandbox is only an approximation of a rover in Martian gravity, atmosphere and soil. Compounding the problem is nature of the Mars launch window. The heavens impose the hardest of deadlines: a mission can only be launched during a short window every 26 months and projects must meet this schedule at all costs. If there are delays in the project's critical path the only alternative is to cut things at the end; fallback modes in which the rover can operate without autonomy are always provided for in case autonomy fails or the schedule slips; these become the mission baseline.

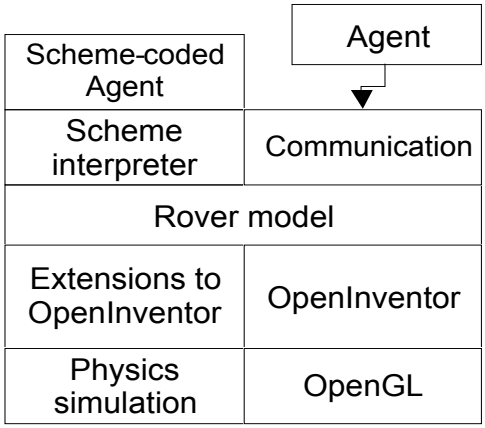
One part of the solution to this problem is to enable autonomy development much earlier in the process, early enough to influence the final design. The purpose of LiveInventor is to provide a software environment in which rovers can be very quickly modeled, and their physical interaction with the world simulated and visualized, at a level of abstraction appropriate for autonomy developers, with accurate masses, joints, actuators, sensors, terrain, gravity and atmospheric conditions.

Another part of the solution is to increase the number of autonomy developers available to work on the problem. LiveInventor has been assembled from open-source and COTS components so that it can be easily and cheaply distributed to academic institutions, enabling professors and students to easily develop software for NASA-relevant challenge problems.

Architecture of LiveInventor

LiveInventor is an application that integrates a physically-based simulation library with a 3-D rendering environment, a scripting language, and a distributed communication system, packaged within a graphical user interface. LiveInventor was built by extending OpenInventor, the graphics library developed by SGI [1]. Inventor models three-dimensional solids using a scenegraph, an ordered acyclic graph in which nodes represent graphics entities or operations. Actions (loading a scenegraph from a file, rendering the

scenegraph, searching it) are accomplished by traversing scenegraphs and changing state as each node is traversed. LiveInventor extends the set of nodes defined by Inventor by introducing nodes that represent kinematic and kinetic parameters like mass and inertia tensor, constraints (joints) between bodies like hinges and springs, geometrical collision models that may differ from the rendered graphical models, and material types and their interactions like friction and collision elasticity. LiveInventor follows OpenInventor's rules for extending the node and state definitions, so LiveInventor nodes behave just like regular OpenInventor nodes, e.g. they are read in with the standard read-from-file command, they are written with the standard write-to-file command, etc. Also, the node definitions can be compiled to a dynamically linked library (DLL) so they can be linked into any existing inventor application.



LiveInventor Architecture

When the LiveInventor nodes are loaded they cause the appropriate dynamic mass objects, constraints, collision models and materials to be created in the physical simulation world. The physics library used in LiveInventor is currently Vortex, sold by Critical Mass Labs.

Autonomy developers can use the embedded Scheme interpreter to communicate with the simulated rover or the environment without having to compile and link C or C++ code. LiveInventor uses the Gambit Scheme library [2]. Gambit is written in C and includes both a Scheme interpreter and Scheme-to-C compiler, making it very easy to call Scheme code from the C++ environment or call C functions from the Scheme environment.

Often autonomy developers have existing systems written in a language other than Scheme, on operating systems other than Microsoft Windows NT, and want to develop and test their code against a simulated rover without having to either recode in Scheme or have to link their C/C++ code into a large executable. Autonomy developers with existing large systems can use Ensemble, an open-source publish-subscribe message passing system from Cornell University to communicate with the simulated rover and its environment [3]. Ensemble is robust, has a small memory footprint, has been ported to most common operating systems (Windows, Mac, most UNIX implementations) and has a simple API.

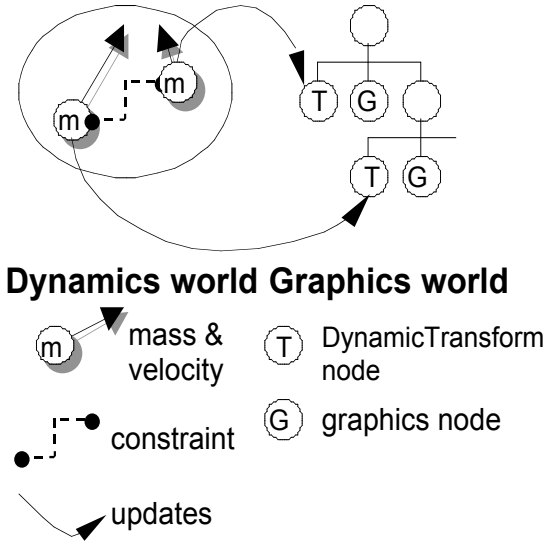
Design Principles

LiveInventor was designed with a number of principles in mind. The first was integrate rather than reinvent, that is, wherever possible use existing software and file formats rather than writing our own. Not only does this save time but it also frees the user from having to learn yet another proprietary language and/or file format. LiveInventor's file format will be quite familiar to anyone familiar with VRML. The second principle was to use open source wherever possible. All components of LiveInventor are open source except for the Vortex simulation engine, and we are looking into developing a version of LiveInventor that uses an open source simulation engine like ODE [4]. Using open source makes redistribution to academic institutions much simpler. A third principle was to make it scalable. We wanted it to be easy for LiveInventor users to throw together a small mechanical simulation, while still being able to simulate and render large, complex environments like the International Space Station. Our fourth principle was portability through portable components. All the individual components comprising OpenInventor run on Linux as well as Microsoft Windows, so porting it to Linux is just a matter of recompiling and linking on Linux.

Under the hood

LiveInventor ties together three different simulation worlds: the dynamics simulation world, the collision detection world, and the graphical world. In OpenInventor 3-D data is represented by a scenegraph data structure. When a scenegraph is rendered, an action traverses the tree and updates global state variables, including the current transformation. Transform nodes when traversed modify the current transformation. LiveInventor extends the transform node by

defining a DynamicTransform node, which represents a body in space and includes information like mass, inertia tensor, initial velocities, etc. in addition to the transformation matrix. When they are created DynamicTransform nodes create a corresponding mass objects (bodies) in the dynamics world. Similarly, collision nodes are extensions of geometry nodes that create corresponding collision object nodes in the collision world when they are loaded. A collision detection node may be associated with a body, in which case its position is tied to the body's position, or not, in which case they act like bodies fixed in space (the floor of the simulated world is usually such a collision node). On each simulation time step, the physics simulation is stepped forward and the mass positions are updated. This new position is propagated to the collision object. A rendering action is then applied to the scenegraph. When this action hits the DynamicTransform node it queries the body in the dynamics world and updates the transform node's position and orientation based on the new position and orientation of the body. The DynamicTransform node is then traversed just like a normal transform node, its transformation matrix being accumulated into the global transform. Constraint (joint) nodes are loaded similarly to DynamicTransform nodes: they create constraint objects connected to bodies in the dynamics world. When the rendering action hits them they do nothing,



unless their graphics representation is turned on (e.g. for a hinge, a lines are drawn to indicate the axis of the hinge and connections to the bodies). Other nodes like ContactParameter nodes that specify friction models between bodies set dynamics world parameters at startup and are ignored during rendering.

Current customers and future directions

LiveInventor is currently being used by the Personal Satellite Assistant project to simulate the interaction of the PSA with astronauts inside the International Space Station. In this scenario LiveInventor must simulate the PSA in real time because it is being controlled by the same software that will ultimately control the hardware PSA. The PSA controller communicates with LiveInventor over CORBA; integration of LiveInventor with the CORBA controller took an afternoon.

LiveInventor is also being used to develop and test diagnostic code for the K9 rover arm. The Model-Based Autonomy group at Ames is developing a system to perform mode and parameter estimation to improve the robustness of rover traverses and instrument placement. They are using a simulation of the K9 robotic arm and a workspace or 'mini Mars yard' surrounding the arm. Its purpose is to increase access to a limited resource (the single arm on K9) and to enable the insertion of faulty components without impacting the main K9 rover. The team's ability to achieve both of these goals can be significantly enhanced through a software simulation of the arm. A simulation can be ready earlier and enable exploration of a greater range of parameter values, workspace configurations, and inserted faults at some cost in fidelity.

Potential directions include large-scale massively parallel simulations to coevolve hardware and controllers, use by mission operations to simulate traversals and tasks to detect potential problems, and integration of physical simulation and modeling into a rover's onboard software path-planning software.

Reference

1. **The Inventor Mentor** by Josie Wernecke, Addison-Wesley Publishing Company, New York: 1994

2. **Gambit-C**, version 3.0, by Marc Feeley. May 1998 http://www.iro.umontreal.ca/~gambit/doc/gambit-c.ps

3. **The Ensemble System,** Mark Hayden. Cornell University Technical Report, TR98-1662, January 1998

4. **Open Dynamics Engine v0.035 User Guide**
Russell Smith,
http://opende.sourceforge.net/ode-latest-userguide.htm

Acknowledgments

This work was performed under NASA contract CSRDS NAS2-00065.