IPSA - Inventor Physical Modelling API for Dynamics Simulation in Manipulation

A. Bierbaum, T. Asfour and R. Dillmann
Institute of Computer Science and Engineering
University of Karlsruhe (TH)
Karlsruhe, Germany
{bierbaum,asfour,dillmann}@ira.uka.de

ABSTRACT

A. Introduction

When studying the dynamic, physical interaction of robots with the environment a generic physics simulator becomes most important when robot and environment exceed a certain complexity. Investigation of current problems in robotics, e.g. in multifingered manipulation or two-legged walking requires simulations which reflect physical details of the real robot system in a virtual scene. The development of suitable control systems for complex manipulators or walking robots in simulation may save a lot of time, otherwise spent for operating the system, and prevents unintented damage on a costly robot system. Further, the application and benefit of using specific sensors may be investigated by physical modelling of a robot system, e.g. when investigating the ideal placement and required resolution of tactile sensors. Finally, model-based simulation allows to distribute models of expensive robot system among researchers, which means saving costs directly. Dynamic simulations have attained focus much earlier in other domains as in development of automotive and space vehicles and for creating realistic computer animations. Consequently, a lot of commercial and non-commercial dynamic simulation frameworks have evolved.

A reference multipurpose modelling and simulation system, not only in robotics, is *Matlab/Simulink* [1]. By using additional extensions to Matlab/Simulink such as *SimMechanics*, the *VR Toolbox* or *The Robotics Toolbox* [2] a lot of mechanisms are provided for simulating robots, their control systems and environmental dynamics. Yet, the assembly of a detailed virtual simulation scene from these building blocks is cumbersome and often takes a considerable portion of time compared to the whole investigation period.

Also for the purpose of developing robot controllers in simulation Microsoft has made its *Microsoft Robotics Studio* (MSRS) available to the public [3]. This IDE provides a comprehensive software framework aiming for rapid development of robot control programs with unified interfaces and deploys the *PhysX* physics engine by NVIDIA for simulation of the virtual scene. Although MSRS appears quite powerful, the construction of a complex scene is not intuitive and requires different models for the physical and the visual domain. Further, the usability of the framework is limited to Microsoft operating systems.

A quite useful robot simulator for developing manipulation and grasp planning algorithms was introduced with *GraspIt!* in [4]. This software provides a robot library with several robot hand models and a robot arm model. The simulator may be invoked via an interactive UI or via a TCP/IP interface for external applications. For static grasp analysis a collision detection function and consideration of friction forces were integrated and used to compute grasp quality measures. Later in [5] a dynamics simulator was added for simulation and investigation of dynamic grasping tasks. Specification of the physics model data such as joint types and location or inertia matrices must be provided by the user in addition to the visualization model. But by the authors practical experience these specifications were hard to tune to achieve a stable simulation behavior.

B. IPSA - Overview

Inventor Physics API (IPSA) is an extension of the Open Inventor toolkit with physics objects using the stable and mature ODE library for simulating rigid body dynamics. IPSA was originated at the Technical University of Braunschweig, Germany in 2004 [1]. We reworked the initial version of IPSA completely to achieve the goal of the originators, which is combining the advantages of an object oriented framework with a hierarchical scenegraph, local reference frames and a rich file exchange format for easily constructing complex 3D physics scenes.

For 3D visualisation we make use of the *Open Inventor Toolkit*, a system independent C++ class library which is used in a range of scientific and engineering 3D visualization systems [2]. Open Inventor provides a fully object oriented API achieving high performance by building on top of OpenGL for rendering. The graphic instances in Open Inventor are 3D Objects which are stored as nodes in a hierarchical scenegraph. The transformation and various

[1]The original version of IPSA used an early release of ODE, but is since then discontinued, see http://www.umi.cs.tu-bs.de/full/education/practical/ss2004/vmp/ipsa/index.html .

[2]Originated by SGI [6], today the commonly used Open Inventor implementation is *Coin3D*, downloadable from www.coin3d.org .

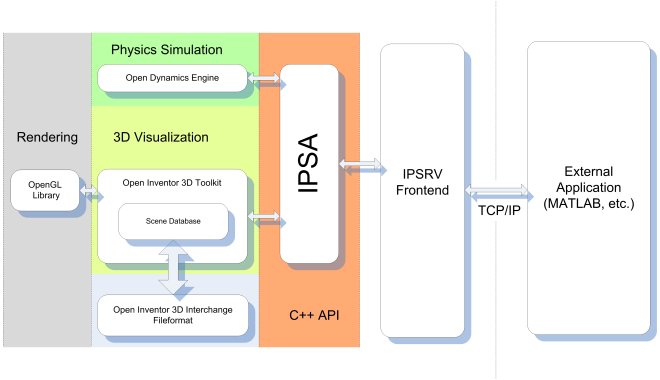

Fig. 1. Software interrelationship between IPSA, IPSRV, external libraries and application.

appearance parameters of an object may be changed in terms of node parameters. Objects can be inserted to or removed from the scene at any time. It is worth to mention that the hierarchical scenegraph concept inherently supports the transparent specification of complex forward kinematics, which is most important for robotics modelling. The toolkit includes dynamic scene functions like object motion and collision checking. An exchange fileformat for importing and exporting a scene graph as a binary or ASCII datafile is provided. The latter allows the specification of a 3D scene as a textfile with a notation similar to XML. Beside the provided classes, the Open Inventor toolkit is extendible with new classes and may thus be customized for special applications [7].

As indicated before, a lot of physics simulation systems have been developed in recent time. A comprehensive overview of publicly available physics engines can be found in [8]. Seven major physics simulation engines have been evaluated here, comparing in particular the performance of the integrator, friction modelling, constraint stability and the collision detector. It was found that no engine performs best at all tasks and thus determining a suitable engine for a specific application is a complex decision for the developer.

For physics simulation we have chosen Open Dynamics Engine (ODE) [9], a freely available physics simulation library written in the C++ programming language. It has become one of the most favored open source physical simulation engines, not only in robotics, as it provides a robust and precise integrator, maintains stable constraints and good documentation. ODE comes with different collision detection routines and also supports the integration of external collission detectors in the simulation loop. For constrained motion various types of joints are provided. Although having a C++-API, ODE is not an object oriented framework and does not support local reference frames, so the development of complex virtual scenes is not simplified. Also, ODE does not offer a file exchange format for importing and exporting scene models. Figure 1 shows the software structure and interrelations between the different libraries involved in the framework. The IPSA class extensions for Open inventor wrap the ODE entities for specifying a physics world in an object oriented way. For rendering, Open Inventor itself makes use of the OpenGL library. The IPSA C++-API may be used to develop a simulation application. Using this API we have created IPSRV as a standalone physics simulator with a TCP/IP interface for controlling and analyzing a simulation by an external application. For most cases in robotics simulation using IPSRV is the most convenient way to develop a particular physics simulation, as only a virtual scene file in the exchange file format must be provided. This scene file may be edited easily by hand using the Open Inventor file format syntax. We will describe both ways of using IPSA in the following sections.

C. Using IPSA

We will present how to use the C++-API for integrating IPSA in a custom simulation application. Further we will show how the IPSRV physics simulator application may easily be used for controlling and analyzing a physics simulation from Matlab/Simulink via a TCP/IP connection.

D. Physical modelling elements in IPSA

In this section we will present the modelling primitives in IPSA, i.e. a physics world, physics bodies and joints. Further we will show how to construct kinematic chains and other elements in a virtual scene. An example of a virtual scene is depicted in figure 2. A major advantage of the

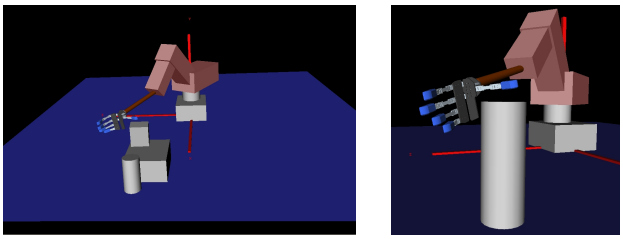

Fig. 2. Example of a virtual scene in IPSA: manipulator arm with anthropomorphic robot hand and objects.

hierarchical scene graph concept using local reference frames is that description in terms of Denavit-Hartenberg-Parameters is not required for specifiying serial link kinematic chains. Rather, a kinematic chain is described as a set of bodies and joints. Joints are specified by the type, which constrains the degrees of freedom of a body. The joint anchor location may be specified in the global *or* in any local reference frame which makes modelling of a robot from data given by an engineering draft very straight forward.

E. Implementing virtual actors and virtual sensors

For practical use of a physics simulation involving robotics, virtual sensor feedback is indispensable. We show how to access force, torque, position and velocity sensing data in the virtual scene from an external client using the IPSRV physics simulator. A visualization example is depicted in figure 3. Also, methods for actuation of joints and bodies are described.

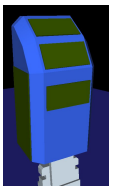

Fig. 3. Finger model with dedicated tactile sensors.

F. Case studies

In this section we will provide several case studies, demonstrating the capabilities and ease of use with IPSA. In particular we will demonstrate constructing a simple physics scene and a several DoF manipulator arm with a robot hand equipped with tactile sensors.

ACKNOWLEDGEMENT

The work described in this paper was conducted within the EU Cognitive Systems project PACO-PLUS (FP6-2004-IST-4-027657) funded by the European Commission.

REFERENCES

[1] The MathWorks, Inc., *Website, The MathWorks - Matlab and Simulink for Technical Computing*, 2008, http://www.mathworks.com.

[2] P.I. Corke, "A robotics toolbox for matlab," *IEEE Robotics & Automation Magazine*, vol. 3, no. 1, pp. 24–32, 1996.

[3] S. Cherry, "Robots incorporated," *IEEE Spectrum*, vol. 44, no. 8, pp. 24–29, 2007.

[4] Andrew T. Miller and Peter K. Allen, "Graspit!: A versatile simulator for grasp analysis," in *Proceedings ASME International Mechanical Engineering Congress & Exposition, Orlando*, Nov. 2000, pp. 1251–1258.

[5] A.T. Miller and H.I. Christensen, "Implementation of multi-rigid-body dynamics within a robotic grasping simulator," in *Proc. IEEE International Conference on Robotics and Automation ICRA '03*, 2003, vol. 2, pp. 2262–2268 vol.2.

[6] SGI, *Open Inventor Standard*, 2007, http://oss.sgi.com/projects/inventor/.

[7] Josie Wernecke, *The Inventor Toolmaker: Extending Open Inventor, Release 2*, Addison Wesley, 1994.

[8] Adrian Boeing and Thomas Bräunl, "Evaluation of real-time physics simulation systems," in *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, New York, NY, USA, 2007, pp. 281–288, ACM.

[9] Russel Smith, *Open Dynamics Engine (ODE), Release 0.9*, http://www.ode.org.