OpenHRP: Open Architecture Humanoid Robotics Platform

Fumio Kanehiro, Hirohisa Hirukawa and Shuuji Kajita

The International Journal of Robotics Research 2004 23: 155
DOI: 10.1177/0278364904041324

Fumio Kanehiro
Hirohisa Hirukawa
Shuuji Kajita

National Institute of Advanced
Industrial Science and Technology
1-1-1 Umezono, Tsukuba,
Ibaraki, 305-8568 Japan

OpenHRP: Open Architecture Humanoid Robotics Platform

Abstract

This paper presents the open architecture humanoid robotics platform (OpenHRP) which consists of a simulator and motion control library of humanoid robots. The binary software developed on OpenHRP can be applied to the real counterpart as is, thank to the proposed hardware abstraction and synchronization mechanism. The compatibility between the simulation and corresponding experiment has been successfully examined. OpenHRP is expected to initiate the exploration of humanoid robotics on open architecture software and hardware.

KEY WORDS—humanoid, HRP, software platform, CORBA, OpenHRP

1. Introduction

For five years, since 1998, the Ministry of Economy, Trade and Industries of Japan has run a Humanoid Robotics Project (HRP; Inoue et al. 2000, 2001). The leader of HRP is Hirochika Inoue from the University of Tokyo. Four copies of a humanoid robot (called HRP-1), a teleoperations cockpit for them and a virtual humanoid robot platform (V-HRP; Nakamura et al. 2000, 2001) have been developed in phase one of HRP as the research platform, and various applications of humanoid robots are under development in phase two on the platform. We call the organization of the project the *platform-based approach*. This is the antithesis of usual robotics projects in which elementary technologies are developed at first and they are integrated into a system at the final stage.

The platform has been made transparent in turn. First, the software of HRP-1 has been developed by Honda R&D as well as its hardware, and it was provided as a black box. Secondly, we have replaced the controller for biped locomotion which was developed on V-HRP. Thirdly, V-HRP has also been replaced by a new simulator on which we can develop portable software to the corresponding hardware. Finally, a new humanoid robot HRP-2 is to be developed, and the controllers examined on HRP-1 will be applied to HRP-2.

The platform employs open architecture software as much as possible. That is, the platform is built on CORBA (common object request broker architecture; Object Management Group 2003) which is a standard middleware, and the real-time controller of the robot is run on ART-Linux (Ishiwata and Matsui 1998) which is a real-time extension of Linux. Besides, the simulator and the controllers have become white boxes whose internal API is clearly described by Interface Definition Language (IDL; Object Management Group 2003). Therefore, we call the package of the simulator and controllers with the compatible humanoid robot OpenHRP, which stands for open architecture humanoid robotics platform.

The unification of the controllers for the simulated and real robot is realized by hardware abstraction and synchronization mechanism and employing ART-Linux in which real-time processing is available at the user level. Thanks to the unification, the controllers can share a significant amount of software with the simulator, including the parameter parser, kinematics and dynamics computation and the collision detector.

The compatibility between the simulation and experiments is enhanced by introducing a new algorithm to simulate contact force and torque between objects. Although the analytical (or impulse-based) methods (Baraff 1989; Cremer and Stewart 1989; Mirtich and Canny 1995) are more numerically stable and efficient than explicit methods (Marhefka and Orin 1996) which compute the contact force and torque, however the explicit methods have an advantage when a mechanical spring–damper is implemented on the feet of a robot such as HRP-1 (Terzopoulos et al. 1987; Witkin, Fleisher, and Barr 1987). The new algorithm consists of a robust geometric algorithm to find the normal vector at each colliding point and a mathematical model to compute the colliding force and torque based on a spring–damper model.

The International Journal of Robotics Research
Vol. 23, No. 2, February 2004, pp. 155-165,
DOI: 10.1177/0278364904041324

Because the unification of the controllers and the compatibility between the simulated and real robots are realized, various fundamental technologies of humanoid robotics can be developed efficiently on OpenHRP. The virtualization of the platform is very important to inherit software library from one hardware to another. For example, Honda R&D took only nine months to replace humanoid robot P2 by P3, since Honda R&D also has a nice virtual platform and most software could be ported with minimum effort through it.

This paper is organized as follows. In Section 2 we introduce the design concept and configuration of OpenHRP with a brief description of HRP-1. In Section 3 we show how the unification of the software on the simulator and that on the real robot has been realized. In Section 4 we describe the applications of OpenHRP, and in Section 5 we show examples of the simulation and experiment. Section 6 concludes the paper.

2. Overview of OpenHRP

2.1. Design Concepts

Although smart humanoid robots demand the integration of various functions including vision, speech recognition and motion control, it is unrealistic for a single institute to develop all of them. The goal of OpenHRP is to provide a research platform on which the users can concentrate on the development of their favorite functions. The design concepts of OpenHRP are as follows.

Modularity. The architecture of the software must be modular to realize the addition and replacement of a single function.

Open architecture. The architecture of the software must be open and built on open architecture software to make users' development easier.

Portability. The software developed on the simulator must be applied to the real robot as is, to avoid possible errors caused by the porting.

The concepts have been realized by the following implementation.

Modularity. OpenHRP is built on CORBA (Object Management Group 2003) which is a standard middleware to integrate object modules.

Open architecture. OpenHRP runs on Linux/Windows and can be compiled by gcc/vc respectively. The models of the robots and objects are defined by VRML97 which is an OSI standard. The signature of each module of OpenHRP is clearly defined by IDL and it is possible for the users to replace it.

Portability. The binary software on the simulator can work on the real robot as is, by introducing the hardware abstraction and synchronization mechanism.

Fig. 1. Configuration of OpenHRP.

2.2. Configuration of OpenHRP

The configuration of OpenHRP is shown in Figure 1. OpenHRP can simulate the dynamics of structure-varying kinematic chains between open chains and closed ones like a humanoid robot (Yamane and Nakamura 1999). It can detect the collision between robots and their working environment including other robots efficiently and precisely on which the forward dynamics of the objects are computed. It can also simulate the fields of vision of the robots, force/torque sensors and inclination sensors according to the simulated motions. We call the simulations *sensor simulations*. The sensor simulations are essential to develop the controllers of the robots. OpenHRP is implemented as a distributed object system on CORBA (Object Management Group 2003). A user can implement a controller using an arbitrary language on an arbitrary operating system if it has a CORBA binding.

The dynamics simulator of OpenHRP consists of five kinds of CORBA servers (see Figure 2) and these servers can be distributed on the Internet and executed in parallel.

Each server can be replaced with another implementation if it has the identical interface defined by IDL. Using the language independence feature of CORBA, ModelParser and OnlineViewer are implemented using Java and Java3D, other servers are implemented using C++. The functions of each server are as follows.

ModelParser. This server loads VRML files describing the geometric models and dynamics parameters of robots and their working environment, and provides these data to other servers.

CollisionChecker. The interference between two sets of triangles is inspected, and the position, normal vector and the depth of each intersecting point are found. RAPID (Gottschalk, Lin, and Manocha 1996) is enhanced to this end.

Dynamics. The forward dynamics of the robots is computed (Nakamura et al. 1999).

Controller. This server is the controller of a robot, which is usually developed by the users of OpenHRP.

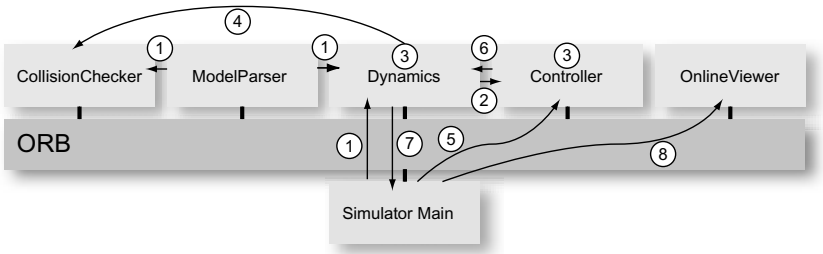

Fig. 2. Dynamics simulator of OpenHRP.

OnlineViewer. The simulation results are visualized by three-dimensional (3D) graphics and recorded.

Using the servers, the forward dynamics of the robots is computed in the following procedure. The total control flow is shown in Figure 2.

Initialization. (1) `ModelParser` reads a VRML file via HTTP protocol. The kinematics and dynamics parameters are sent to `DynamicsServer` and the geometric model is to `CollisionChecker`.

Computation. (2) `Controller` reads the outputs of the simulated sensors while communicating with `DynamicsServer`. (3) `Controller` and `DynamicsServer` finds the torques of the actuators and the updated states respectively. Note that these servers can be run in parallel. (4) While the forward dynamics is computed, `CollisionChecker` is called to find the updated position, normal vector, and the depth of each intersecting point. (5) The client sends the updated states to `Controller`, and (6) `Controller` sends the outputs to `DynamicsServer`.

Visualization and recording. (7) The client gets the current states from `DynamicsServer`,and (8) send them to `OnlineViewer` which visualizes and records the current states.

The client is the scheduler of the simulation and `OnlineViewer` displays the results using 3D graphics in Figure 2. The users can also use the integrated simulation environment (ISE) instead. The users can set the initial states of the simulation and specify the pairs of the objects for the collision checking. The results of the simulation including the joint trajectories and the contact force can be displayed on ISE synchronized with 3D graphics. A snapshot of ISE is shown in Figure 3.

The biped locomotion of a humanoid robot with 29 degrees of freedom (DOFs) was simulated to evaluate the performance of the dynamics simulator. The collision check was applied to the pairs between all the links of the robots and the ground. The forward dynamics computation of the robot took 25 ms for each integration interval on Intel Pentium III 933 MHz with memory of 512 MB and OS:Linux Ver. 2.2.17.

2.3. Humanoid Robot HRP-1S

As an example of compatible humanoid robots with the simulator, humanoid robot HRP-1 is investigated in this paper. The hardware configuration of the controller of HRP-1 is shown in Figure 4. The humanoid robot is called HRP-1S in the following to clarify that the control software of the robot is not provided by Honda but developed by us. The real-time controller runs on ART-Linux. The internal states of the controller can be logged at an external PC through the reflective memory connected by an optical fiber.

3. Implementation of OpenHRP

3.1. Unification of the Applications

The applications developed on the simulator can be ported as is to the real counterpart. This concept is illustrated in Figure 5. The portability of the applications on OpenHRP between the simulator and the real robot is realized by three tricks: the employment of ART-Linux, hardware abstraction and the sychronization mechanism.

3.1.1. Employment of ART-Linux

It is not possible to realize the portability on a real-time operating system such as VxWorks on which applications are compiled by a cross-compiler on a different operating system. Although it may be possible to realize the compatibility at the source level, the development process is not efficient then. Besides, rich development tools are not available on VxWorks.

RT-Linux (Yodaiken and Barabanov 2003) is another candidate for our purpose, but real-time tasks can be implemented as kernel modules on RT-Linux and then a limited library can be used from the real-time tasks. Besides, a special mechanism called RT-FIFO is used for the communication between the kernel and user space, therefore the mechanism must be virtualized as well to realize the portability.

ART-Linux is a real-time extension of Linux on which real-time tasks can be executed in the user space. Thanks to the feature, real-time tasks can utilize most library on Linux, and

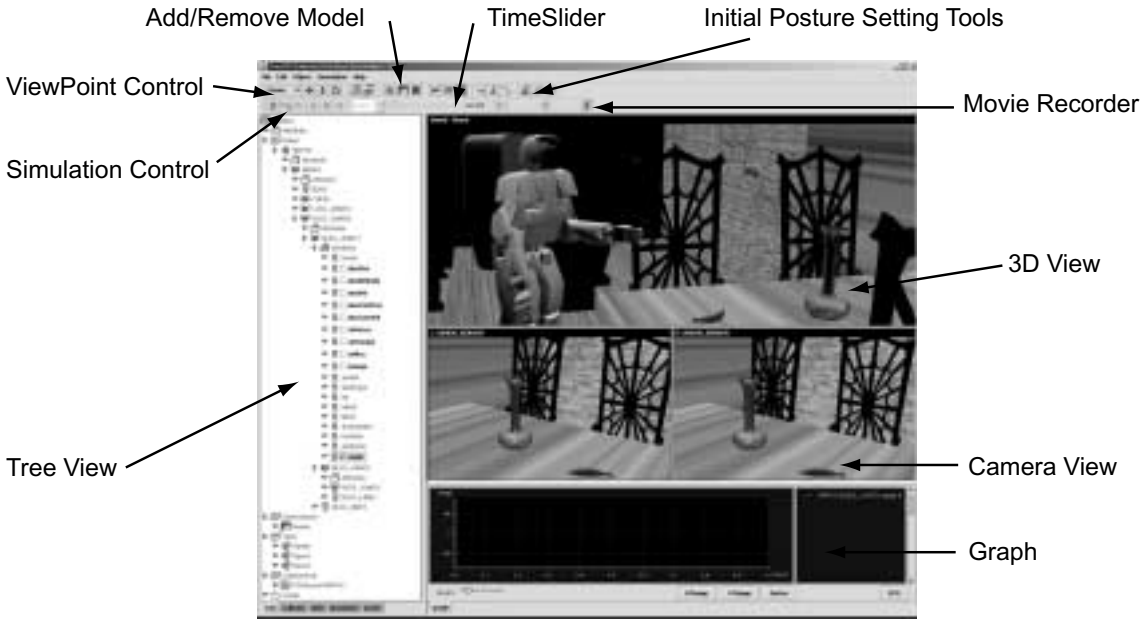

Fig. 3. Integrated simulation environment.

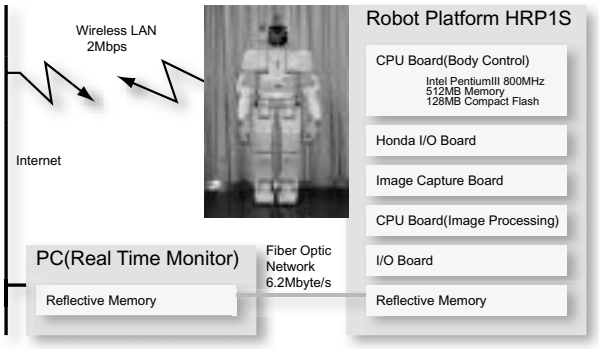

Fig. 4. Controller hardware of HRP-1S.

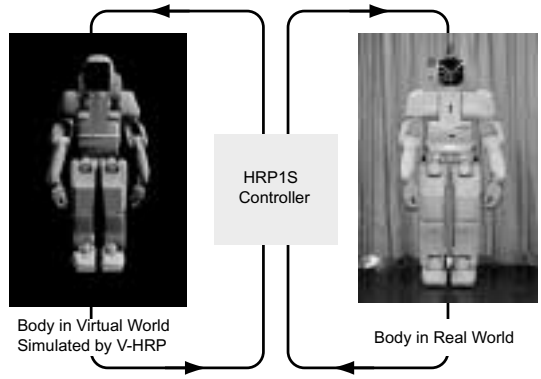

Fig. 5. Unification of the applications on OpenHRP.

the portability can be realized without the virtualization of RT-FIFO.

3.1.2. Abstraction of the Hardware

The application programming interface (API) of the virtual robot on the simulator is not identical to that of the real robot in general. This is the reason why some porting task is demanded. OpenHRP introduces an abstracted API, and uses an emulation adaptor on the API of the virtual robot and a hardware adaptor on that of the real robot to absorb the difference of APIs (Kanehiro et al. 2001a, 2002b). This mechanism is illustrated in Figure 6.

The abstracted interface consists of four APIs: `open()`, `read()`, `write()` and `close()`. `open()` establishes a connection to the robot and initialize it. `read()` reads the outputs of the sensors periodically. `write()` outputs the commands to the robot. `close()` terminates the connection to the robot and cleans it up. These APIs are implemented as a C++ class, and the adaptors inherit this class and must be implemented according the specifications of the robot. The definition of the abstracted interface is as follows.

```
class robot_adaptor
{
public:
  virtual bool open(int argc, char *argv[]);
  virtual bool close();
  virtual bool read(robot_state *rs);
  virtual bool write(motor_command *mc);
};
```

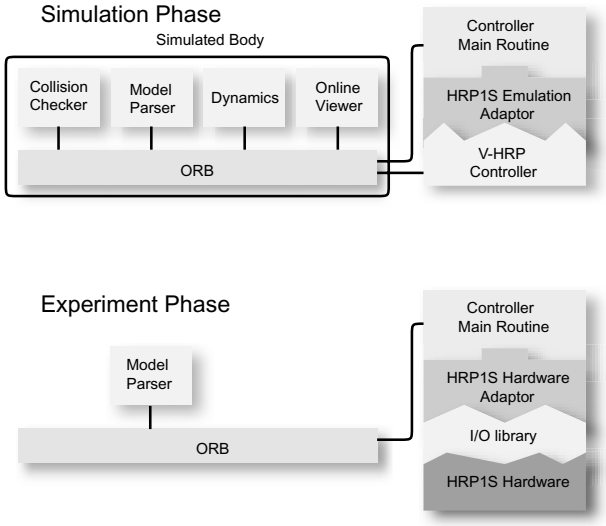

Fig. 6. Abstracted API and the adaptors.

The hardware adaptor is implemented by *art_enter*, *art_wait* and *art_exit* which are system calls of ART-Linux for real-time tasks. `open()` calls *art_enter* to make a process into a real-time one. `read()` calls *art_wait* to obtain the outputs of the sensors periodically, and `write()` to send the commands to the robot. `close()` calls *art_exit* to restore the real-time process to a non-real-time process.

We can switch between the simulation and the experiment by replacing the adaptor. The adaptors are given by shared objects which can be linked dynamically according to the simulation or the experiment.

3.1.3. Synchronization Mechanism

The time passes at a different speed in the simulated and real worlds. In general, the forward dynamics simulation takes longer than the corresponding real time. So it must be synchronized to realize the binary compatibility. To this end, the time in the simulation is updated when an integration interval for the forward dynamics computation is terminated, and that in the real world proceeds when `read()` calls *art_wait*.

3.2. Isolation of ORB

The controller and the dynamics simulator can share a significant amount of codes. For example, collision detection is one of the major building blocks of the simulator, and it is also essential in the controller to avoid the self-collision of a moving humanoid robot. It is needless to say that basic vector and matrix operations are included in both the simulator and the controller. The parameter parser can also be shared. The forward kinematics computation of robots is also common. The unified controller makes the code sharing easier, and therefore the development of the controllers more efficient. More good news about the code sharing is that the controllers can be more reliable since the building blocks borrowed from the simulator have been already been examined intensively by the simulation.

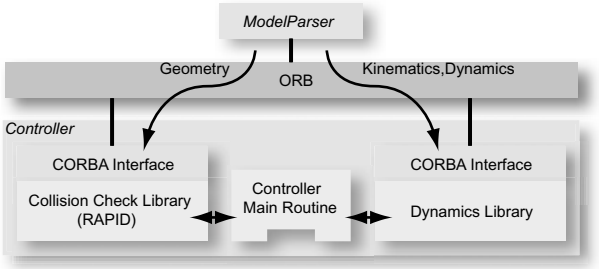

Fig. 7. Internal configuration of the controller.

However, there is a barrier to reuse the code in the controller. That is, the servers such as `CollisionChecker` or `DynamicsServer` are implemented as CORBA servers, as shown in Figure 1. Although real-time functions have been included in the specifications of CORBA since version 2.4 and we can find the implementations of the functions, the overhead of the Internet Inter-Orb Protocol (IIOP) used in CORBA is not small enough for the controller which must update the outputs at a few milliseconds.

This overhead can be bypassed by the following architecture. Let the servers such as `CollisionChecker` or `DynamicsServer` consist of two layers. The lower layer is a normal library which is independent of CORBA, and the higher layer wraps the library by CORBA interface and converts the data structures between the library and the interface. For example, these servers call `ModelParser` through the ORB when reading a VRML file describing the parameters of robots, but they access non-CORBA interfaces when they control the robots in real time. The internal configuration of the controller is shown in Figure 7.

3.3. Determining the Contact Force

The unification of the applications on the simulator and the real robot does not make sense if the simulation is not compatible with the corresponding experiment. A new algorithm was developed to have a better compatibility especially when a humanoid robot has a mechanical spring–damper at its feet. The algorithm is outlined as follows.

3.3.1. Finding the Interference State

We have proposed a complete algorithm to find the constraints which is imposed on a polyhedral object in contact with another (Hirukawa, Matsui, and Takase 1994). The key idea to

determine the constraints is finding separating planes between the neighborhoods of a contact point in two objects. However, there are two reasons why the complete algorithm does not work to find the interference state in the dynamics simulation, as follows.

- `CollisionChecker` detects the interference between two sets of triangles which are obtained from the decomposition of the boundary representation of two objects under consideration. Then the topological information between the triangles is lost, and it is not possible to analyze the neighborhoods for finding the separating planes.

- The objects may penetrate each other during the numerical dynamics simulation, and therefore there is no longer a separating plane between the neighborhoods.

The first problem is solved by recovering the topological data to find the neighborhoods, which can be done in $O(1)$ time using the OBB-tree structure (Gottschalk, Lin, and Manocha 1996). There is no way to avoid the second one, when we simulate the dynamics on a floating point arithmetic. Therefore, the only option for us is to find a "reasonable" constraint at each intersecting point. We choose the constraint that gives the smallest penetration depth in the neighborhood of each penetrating point.

3.3.2. Finding the Contact Force

When the interference state is determined, the next mission is to find the contact force from it. The outline of the algorithm is as follows.

1. From the collection of the penetrating points, estimate the infinitesimal penetration depth between two objects by the least-squares method.

2. Apply the depth to a virtual spring model of the mechanism, and compute the corresponding force.

3. Determine the penetrating velocity from the contact constraints (Ohwovoriole and Roth 1981).

4. Apply the velocity to a virtual damper model (Hunt and Crossley 1975) of the mechanism, and compute the corresponding force.

5. Output the sum of the spring force and the damper force.

4. Applications of OpenHRP

We have developed several applications on OpenHRP which are introduced as follows.

4.1. Real-Time Collision Checker

When a humanoid robot is moving, it is desirable that the self-collision between the links of the robot is checked in real time for enabling emergency stopping of the robot. To this end, a real-time collision checker has been developed. This is also an example of the code sharing between the simulator and the controller.

Let N be the number of the links of a humanoid robot. Then the number of pairs of the links is $\binom{N}{2}$, and the collision detection must be executed for $\binom{N}{2}$ pairs. Taking into account that the interference between consecutive links does not occur since the movable range of the joints are limited, we still need to check $\binom{N}{2} - (N-1)$ pairs. The number of this combination becomes 350 in the case of HRP-1S for which $N = 29$. The average computation time for the collision detection is about 10 ms, when HRP-1S takes many random postures. Here, the geometric model of HRP-1S consists of about 10,000 polygons. This is not fast enough, because the cycle of the controller is 5 ms and we assume that 20% of it can be assigned for the collision inspection. From the above experiment, about 35 pairs can be checked in 1 ms.

Then we propose the following approach combining an off-line pre-processing and a reduced on-line computation.

1. Check the self-collision at a rough resolution for each joint, and try to find safe range for some joint.

2. Re-check the self-collision at a fine resolution for the same range found at the previous step.

3. While keeping the joint of the arms within the safe region, check the self-collision in real time only for the suspicious pairs.

In the case of HRP-1S, we need not check the collisions between the arm and the leg on the same side if we keep the pitch joint of the shoulder or the roll joint in the safe range.

There is no significant safe range for the collision between the left and right legs, but each roll link of the hip and each pitch link of ankles are covered by the other links and therefore need not be checked.

We assume that only legs are moving during walking, 16 pairs must be checked between the legs, 16 pairs between two links around the hands and the legs, four pairs between the main body and foot links, and 36 in total, which can be done in real time. An example posture while walking is shown in Figure 8. An experimental result is shown in Figure 9. The required time is longest when the robot is standing, and decreases while walking.

4.2. Walking Pattern Generator

A walking pattern generator is part of the controller which manages a dynamically stable biped walking. The walking pattern generator was also developed by utilizing the shared codes with the simulator as well as the collision checker. At

Fig. 8. Safe posture for walking.

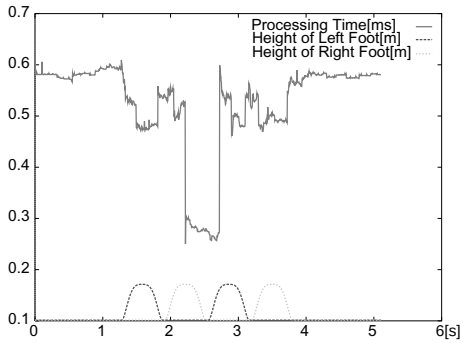

Fig. 9. Computation time for the self-collision while walking.

first, the pattern generator obtains the mass, the center of mass and other parameters for each link of the robot by the model parser. This information is used to calculate and to control the location of the center of mass of the whole robot.

During walking, the dynamics of a biped robot can be approximated by a single inverted pendulum which connects the supporting foot and the center of mass of the whole robot. However, even with this approximation, the inverted pendulum has vast possibilities of moving pattern which are not good for walking. To pick up the suitable motion for walking we constrain the center of mass to move on a plane specified by

$$z = k_x x + k_y y + z_c \tag{1}$$

where (x, y, z) is the position of the mass with respect to the supporting point, $(k_x, k_y, -1)$ specifies the normal vector of the constraint plane and z_c is the z intersection. In the case of the walk on a flat floor, the constraint plane is horizontal and the height of the center of mass is kept constant.

We obtain the following dynamics of the pendulum under the constraints

$$\ddot{x} = \frac{g}{z_c} x + \frac{1}{m z_c} u_p, \tag{2}$$

$$\ddot{y} = \frac{g}{z_c} y - \frac{1}{m z_c} u_r. \tag{3}$$

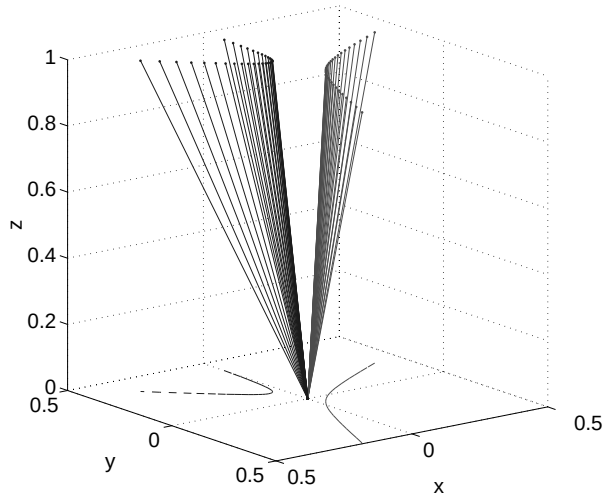

Fig. 10. 3D linear inverted pendulum mode.

We call this dynamics the 3D linear inverted pendulum mode (3D-LIPM; Kajita et al. 2001a). The only parameter which governs 3D-LIPM is z_c, i.e., the z intersection of the constraint plane and the inclination of the plane never affects the horizontal motion. Since eqs. (2) and (3) are linear and independent, we can easily obtain the closed-form solutions which can be directly used for a dynamic biped walking. Particularly, when the input torques of the supporting point are zero ($u_r = u_p = 0$), we obtain hyperbolic trajectories on the constraint plane (Figure 10).

The reference joint angles and speeds are calculated by the inverse kinematics so that the position and the velocity of the feet with respect to the center of mass follow the 3D-LIPM. For this part of implementation, again, we intensively used the shared code from `DynamicsServer` to develop the pattern generator efficiently and reliably. Figure 11 shows the snapshots of the generated walking by our method.

With the benefit of the linear dynamics of eqs. (2) and (3), the pattern generator can create the joint angles and the velocity within 1 ms. To evaluate the walking pattern generator, we also conducted an experiment controlling a 12-DOF biped robot by using a game pad (Kajita et al. 2001b).

4.3. More applications

More applications were developed on OpenHRP including:

(1) feedback controller for biped locomotion;

(2) balance controller at the standing position;

(3) collision avoidance planner for arm motions.

(1) tries to stabilize the biped locomotion of the robot (Yokoi et al. 2001), (2) keeps the balance of the robot which is sup-

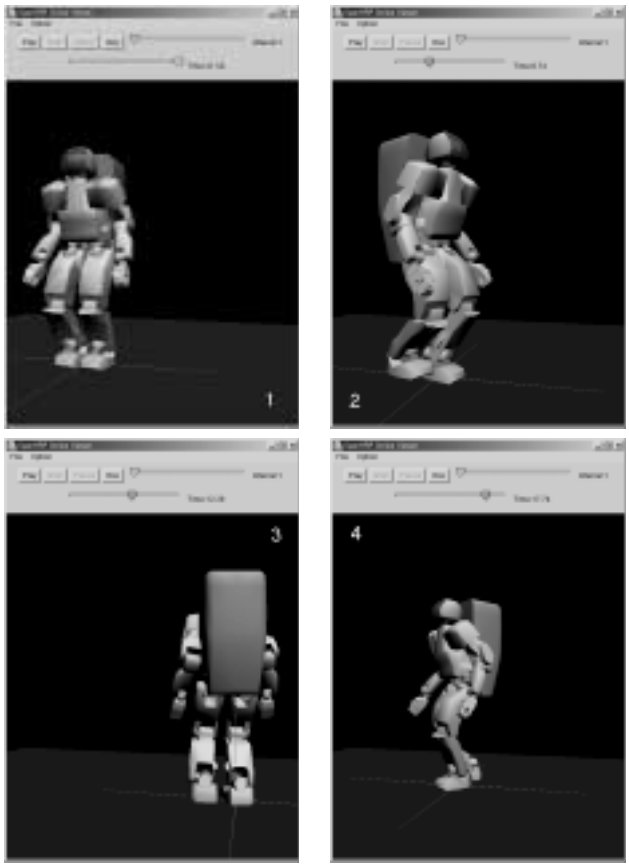

Fig. 11. Snapshots of the generated walking pattern.

posed to carry out some task at the standing position (Nakamura et al. 2000), and (3) plans collision avoidance motions of an arm of the robot (Nakamura et al. 2000).

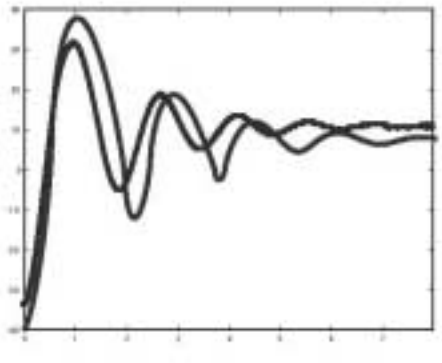

Fig. 12. Damped oscillation of HRP-1S.

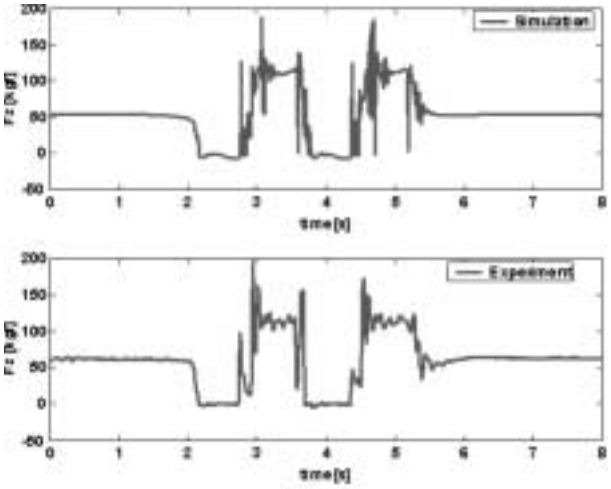

Fig. 13. Normal repulsive force from the floor.

5. Examples

The dynamics simulation and the corresponding experiments are carried out to examine whether the proposed algorithms can realize the consistency between the simulation and the experiment. The first example is a damped oscillation of the robot. The robot is inclined slightly at the standing position, and the external force to incline the robot is removed. Then the robot shows a damped oscillation along the pitch axis, when no feedback control is applied. Figure 12 shows the torque along the pitch axis of a foot of the robot. The curve with the higher first peak is the result of the simulation, and the other is that of the experiment. The cycle of the oscillation and the attenuation rate seems to coincide sufficiently. The curves converge to slightly different values, which is caused by the calibration error to find the posture of the robot with the zero inclination.

This experiment is a good way to tune the parameters of the virtual spring–damper. In the example, the parameters were identified from the physical parameters of the spring–damper mechanism, and fortunately the identified parameters work well. When the parameters are no good, we can tune the parameters experimentally. This identification process is very important to keep the consistency.

The next example is the comparison of the normal repulsive force from the floor to a foot of the robot while it is walking with a feedback control. In Figure 13 the top curve is from the simulation and the bottom curve is from the experiment. The curves also seem to coincide sufficiently. Figure 14 shows the corresponding curve when the repulsive force is found by an impulse based method in V-HRP (Nakamura et al. 2000). The robot seems to oscillate at a high frequency, since the simulation is based on a rigid object model. It is very difficult to evaluate feedback laws on such a simulation. This is one of the major features of the proposed algorithm.

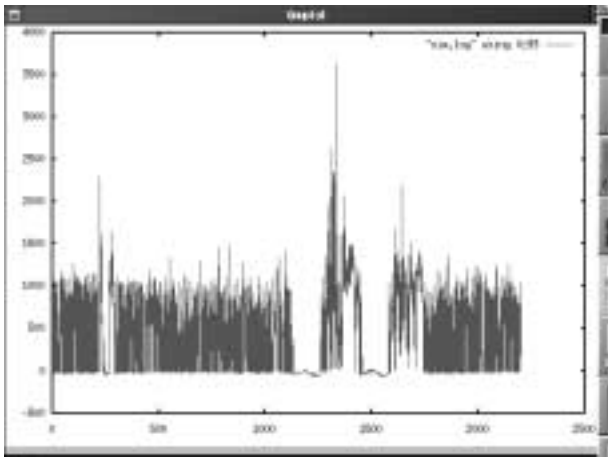

Fig. 14. Normal repulsive force by a rigid object model.

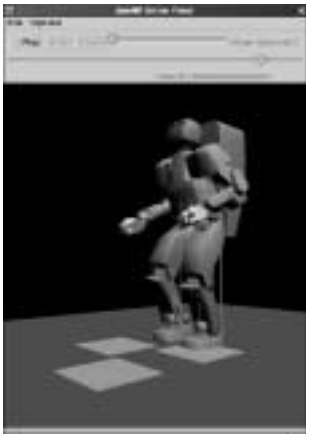

Fig. 15. Walking on a rough terrain.

Figure 15 shows a snapshot while the robot is walking on a rough terrain. The proposed geometric algorithm makes it possible to execute such a simulation in which the colliding points and the corresponding normal vector should be found robustly. Figure 16 shows the snapshots of the walking simulation of another humanoid robot, HRP-2P, and Figure 17 those of the experiment.

6. Conclusions

In this paper we have presented an open architecture humanoid robotic platform, OpenHRP. The results can be summarized as follows.

- OpenHRP is an open architecture platform on which a user's function can be developed efficiently as a module. The CORBA servers of OpenHRP can also be replaced easily by a user.
- The unification of the controllers for the virtual and real robot has been realized by introducing the adapters for two robots respectively with the synchronization mechanism, and employing ART-Linux on which real-time processing is available at the user level.
- The consistency between the simulator and the robot is enhanced by the new algorithm to simulate a general spring–damper mechanism.
- Thanks to the unification, the controllers can share softwares with the dynamics simulator including the parameter parser, kinematics and dynamics computations and the collision detector. This feature can make the development of the controllers more efficient and the developed controllers more reliable. A real-time collision checker and a walking pattern generator for humanoid robots have been developed as examples of the shared code.

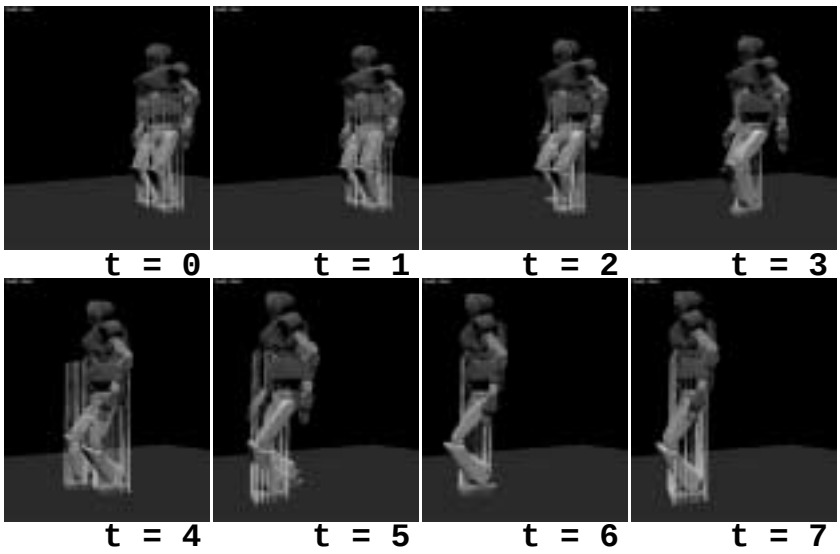

Fig. 16. Walking HRP-2P in the simulation.

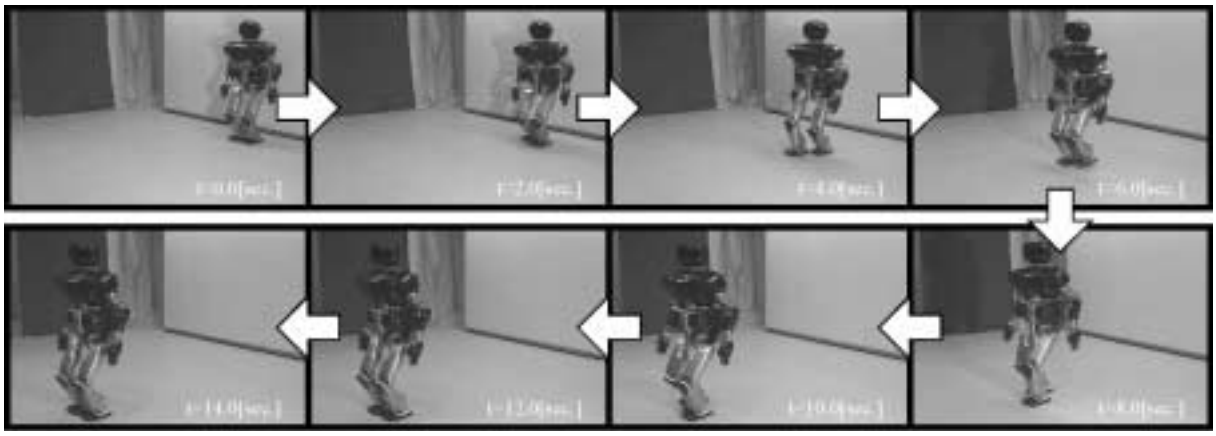

Fig. 17. Walking HRP-2P in the experiment.

We claim that a humanoid robotics platform deserves to be called a platform if the identical software can be used either on the simulator or on the robot and if the consistency between them is satisfactory kept. We believe that OpenHRP is expected to be the first humanoid robot platform in this sense and that it can initiate the exploration of humanoid robotics on open architecture hardware and software.

Acknowledgments

This research was supported by the Humanoid Robotics Project of the Ministry of Economy, Trade and Industry. The authors thank Hirochika Inoue, Yoshihiko Nakamura and Katsu Yamane from the University of Tokyo, and Kazuhito Yokoi, Kenji Kaneko and Kiyoshi Fujiwara from AIST for their cooperation.

References

Baraff, D. 1989. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Computer Graphics* 23(3):223–232.

Cremer, J. F., and Stewart, A. J. 1989. The architecture of Newton, a general-purpose dynamics simulator. In

Proceedings of the IEEE International Conference on Robotics and Automation, Arizona, USA, pp. 1806–1811.

Gottschalk, S., Lin, M. C., and Manocha, D. 1996. OBB-Tree: a hierarchical structure for rapid interference detection. In *Proceedings of ACM Siggraph'96*, New Orleans, LA, USA.

Hirukawa, H., Matsui, T., and Takase, K. 1994. Automatic determination of possible velocity and applicable force of frictionless objects in contact from a geometric model. *IEEE Transactions on Robotics and Automation* 10(3):309–322.

Hunt, K. H., and Crossley, F. R. E. 1975. Coefficient of restitution interpreted as damping in vibroimpact. *ASME Journal of Applied Mechanics* 42:440–445.

Inoue, H. et al. 2000. HRP: Humanoid Robotics Project of MITI. In *Proceedings of the 1st IEEE-RAS International Conference on Humanoid Robots*, Massachusetts, USA.

Inoue, H. et al. April 2001. Overview of Humanoid Robotics Project of METI. In *Proceedings of the 32nd ISR*, Seoul, Korea.

Ishiwata, Y., and Matsui, T. 1998. Development of Linux which has advanced real-time processing function. In *Proceedings of the 16th Annual Conference of the Robotics Society of Japan*, Hokkaido, Japan, pp. 355–356.

Kajita, S., Kanehiro, F., Kaneko, K., Yokoi, K., and Hirukawa, H. 2001a. The 3D Linear Inverted Pendulum Model: A simple modeling for a biped walking pattern generation. In *Proceedings of the 2001 IROS*, Maui, Hawaii, pp. 239–246.

Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Yokoi, K., and Hirukawa, H. 2001b. A real-time pattern generator for biped walking. In *Proceedings of the 2002 ICRA*, Seoul, Korea, pp. 31–37.

Kanehiro, F., Inaba, M., Inoue, H., Hirukawa, H., and Hirai, S. 2001a. Developmental software environment that is applicable to small-size humanoids and life-size humanoids. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, Seoul, Korea.

Kanehiro, F. et al. 2001b. Virtual humanoid platform to develop controllers of real humanoid robots without porting. In *Proceedings of the 2001 IEEE IROS*, Seoul, Korea.

Marhefka, D. W., and Orin, D. E. 1996. Simulation of contact using a nonlinear dampling model. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Minnesota, USA, pp. 1662–1668.

Mirtich, B., and Canny, J. 1995. Impulse-based dynamic simulation. In *Algorithmic Foundations of Robotics*, A K Peters, Natick, MA, pp. 407–418.

Nakamura, Y., Yamane, K., Hirukawa, H., and Nagashima, F. 1999. Distributed computation environments that accelarate humanoid robotics research and applications, *Preprints of ISRR*.

Nakamura, Y. et al. 2000. V-HRP: virtual humanoid robot platform. In *Proceedings of the 1st IEEE-RAS International Conference on Humanoid Robots*, Massachusetts, USA.

Nakamura, Y., Hirukawa, H., Yamane, K., Kajita, S., Fujiwara, K., Kanehiro, F., Nagashima, F., Murase, Y., and Inaba, M. April 2001. Humanoid robot simulator for the METI HRP project. In *Proceedings of the 32nd ISR*, Seoul, Korea.

Object Management Group. 2003. http://www.omg.org/.

Ohwovoriole, M. S., and Roth, B. 1981. An extension of screw theory. *Transactions of the ASME, Journal of Mechanical Design* 103:725–735.

Terzopoulos, D., Platt, J., Barr, A., and Fleisher, K. 1987. Elastically deformable models. *Computer Graphics* 21(4):205–214.

Witkin, A., Fleisher, K., and Barr, A. 1987. Energy constraints on parameterized models. *Computer Graphics* 21(4):225–232.

Yamane, K., and Nakamura, Y. 1999. Dynamics computation of structure-varying kinematic chains for motion synthesis of humanoid. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, Michigan, USA, pp. 714–721.

Yodaiken, V., and Barabanov, M. 2003. *RT-Linux*, http://www.fsmlabs.com.

Yokoi, K., Kanehiro, F., Kaneko, K., Fujiwara, K., Kajita, S., and Hirukawa, H. 2001. A Honda humanoid robot controlled by AIST software. In *Proceedings of the 2nd IEEE-RAS International Conference on Humanoid Robots*, Tokyo, Japan.