Proceedings of the 2001 IEEE/RSJ
International Conference on Intelligent Robots and Systems
Maui, Hawaii, USA, Oct. 29 - Nov. 03, 2001

Virtual Humanoid Robot Platform to Develop Controllers of Real Humanoid Robots without Porting

Fumio Kanehiro[1] Natsuki Miyata[1] Shuuji Kajita[1] Kiyoshi Fujiwara[1] Hirohisa Hirukawa[1]
Yoshihiko Nakamura[2] Katsu Yamane[2] Ichitaro Kohara[3] Yuichiro Kawamura[4] Yoshiyuki Sankai[4]
[1]National Institute of Advanced Industrial Science and Technology, METI.
1-1-1 Umezono, Tsukuba, Ibaraki, 305-8568 Japan
[2]The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8656 Japan
[3]Manufacturing Science and Technology Center, 1-2-2 Atago, Minato-ku, Tokyo, 105 Japan
[4]University of Tsukuba, Tennodai, Tsukuba, 305-0006 Japan

Abstract

This paper presents a virtual humanoid robot platform (V-HRP for short) on which we can develop the identical controller for a virtual humanoid robot and its real counterpart. The unification of the controllers for the virtual and real robot has been realized by introducing software adapters for two robots respectively and employing ART-Linux on which real-time processing is available at the user level. Thanks to the unification, the controllers can share softwares with the dynamics simulator of V-HRP, 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.

1 Introduction

In order to prevent the possible damages of a robot and its working environment during the development process of its controllers, the controllers must be examined on a simulator of the robot first and then applied to move the real robot. The verification of the controllers for humanoid robots is more crucial, since the robot may tip over when it loses the balance. However, the porting of the controller is not straightforward if the run-time environment for the virtual and real robots are significantly different, and the non-trivial porting may induce troubles when it is applied to the real robot.

This paper presents a virtual humanoid robot platform (V-HRP for short) on which we can develop the identical controller for a virtual humanoid robot and its real counterpart. This concept is illustrated in Fig.1. The unification of the controllers for the virtual and real robot has been realized by introducing software adapters for two robots respectively and employing ART-Linux on which real-time processing is available at the user level. Thanks to the unification, the controllers can share softwares with the dynamics simulator of V-HRP[1] developed in METI's Humanoid Robotics Project (HRP for short)[2], including the parameter parser, kinematics and dynamics computations and the collision detector.

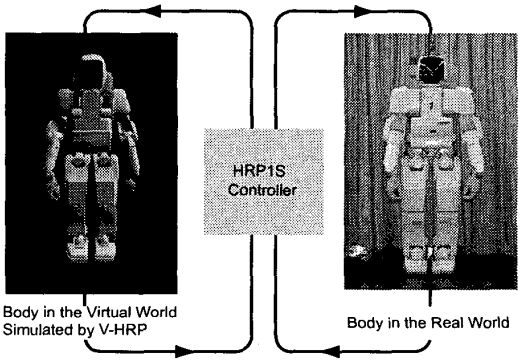

Figure 1: Shared controller for virtual and real robots

This feature can make the development of the controllers more efficient and the developed controllers more reliable. Besides, it becomes easier to feedback the experimental results for the improvements of the dynamics simulator when the run-time environments of the controllers are identical.

This paper is organized as follows. Section 2 overviews V-HRP and humanoid robot HRP-1S developed by Honda R&D as the real counterpart. Sec-

0-7803-6612-3/01/$10.00©2001 IEEE

tion 3 shows how the controllers for the virtual and real robot is unified. Section 4 presents a realtime collision detector for humanoid robots as an example of shared codes in the controller. Section 5 concludes the paper.

2 Virtual and Real Humanoid Robots

2.1 Virtual Humanoid Robot Platform

2.1.1 Overview of V-HRP

V-HRP is a software platform to develop software for humanoid robots including the controllers and has the following features.

- It can simulate the dynamics of structure-varying kinematic chains between open chains and closed ones like humanoid robots[3].
- It can detect the collision between robots and their working environment including other robots very fast and precisely on which the forward dynamics of the objects are computed.
- It can simulate the fields of vision of the robots, force/torque sensors and gradient sensors according to the simulated motions. We call the simulations *sensor simulations*. The sensor simulations are essential to develop the controllers of the robots.
- It is implemented as a distributed object system on CORBA(Common Object Request Broker Architecture)[4]. A user can implement a controller using an arbitrary language on an arbitrary operating system if it has a CORBA binding.

2.1.2 Configuration of V-HRP

V-HRP consists of five kinds of CORBA servers and theses servers can be distributed on the Internet and executed in parallel. Each server can be replaced with another implementation if it has the same interface defined by IDL (Interface Definition Language). Using the language independency 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 a VRML file 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[5] is enhanced to this end.

Dynamics The forward dynamics of the robots are computed.

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

OnlineViewer The simulation results are visualized by 3D graphics and recorded.

2.1.3 Description of Models

The models of robots and their working environment are described by VRML97 format that is extended for a humanoid animation by h-anim working group[6]. The extended version is called *h-anim format*, which includes the definitions of geometry, kinematics and dynamics parameters required for the dynamics computation. An example of h-anim format is as follows.

```
DEF HRP1 Humanoid {
  humanoidBody [
    DEF WAIST Joint {
      jointType "free"
      translation 0 0 0
      rotation 0 0 1 0
      children [
        DEF BODY Segment {
          mass 0.5
          momentsOfInertia [1 0 0
                            0 1 0
                            0 0 1]
          children [
            Inline {url "shape.wrl"}
          ]
        }
        DEF LEG_JOINT0 Joint {
          jointType "rotate"
              ....
```

Here, `Humanoid` node expresses the whole body of a humanoid robot, `Joint` node defines the type of a joint, and `Segment` node describes the parameters of a link, including the mass, the moments of inertia, and the center of the mass.

2.1.4 Execution of The Simulation

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

Setting up of the simulation environment (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`.

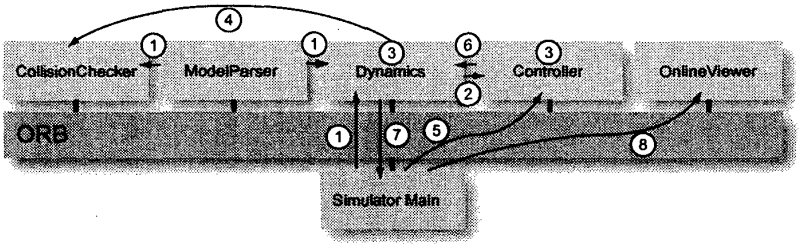

Figure 2: V-HRP Overview

Execution of the dynamics simulation (2) `Controller` reads the outputs of the simulated sensors while communicating with `DynamicsServer`. (3) Let `Controller` and `DynamicsServer` execute the computations. Note that these computations can be run in parallel. The outputs of `Controller` are the torques of the actuators, and those of `DynamicsServer` are the updated states of the robot. (4) While the forward dynamics is computed, `CollisionChecker` is called to find the position, normal vector, and the depth of each intersecting point. (5) After these computations, let `Controller` send the control outputs to `DynamicsServer`. (6) `Controller` sends the outputs to `DynamicsServer`.

Visualization and recording (7) Acquire the current states of the world from `DynamicsServer`. (8) Send them to `OnlineViewer` which visualizes the simulated world and records it.

2.1.5 Performance Evaluation

In order to evaluate the performance of V-HRP, biped locomotion of a humanoid robot is simulated. The sample humanoid robot has 6DOF arms, 6DOF legs, 3DOF waist, 2DOF neck and 29DOF in total. The interference between every links of the humanoid and the ground is checked and the interference between a foot link and the ground always occurs during the simulation. The specifications of the used computer include CPU:Intel PentiumIII 933MHz, Memory:512MB, and OS:Linux-2.2.17. The computation time except the visualization is 25[ms] per the unit integration time, which is usually set around 1 [ms].

2.2 Humanoid Robot HRP-1S

The configuration of the controller hardware of humanoid robot HRP-1S is shown in Fig.3. The real-time controller runs on a CPU board in the backpack

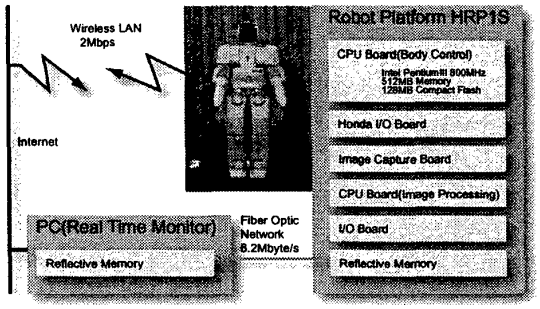

Figure 3: Controller hardware of HRP-1S

of HRP-1S, whose operating system is ART-Linux[7]. ART-Linux enables the execution of realtime processes at the user level, while RT-Linux[8] realizes it only at the kernel level. Thanks to this feature of ART-Linux, users can implement realtime applications as if they are non-realtime ones. This is the first key to realize the identical controller for the virtual and real robot.

Using the optical fiber network connecting the pair of the reflective memory mounted on the backpack and the PC outside the robot, the internal states of the robot can be monitored at the PC in realtime.

3 Unification of The Controllers

The controller must be implemented by exactly same methods with the same signatures for the virtual and real robots, to realize the unification of the controllers; the following two requisites must be hold to this end. (1) abstraction of the controller API where the software body looks like the same as the hardware body, and (2) a sychronizing mechanism that can absorb the difference between the speed of time in the simulated and the real world.

3.1 Hardware Abstraction

The first requisite is realized based on the plug-in architecture [9], where application software is separated into two layers at the adapter level and the counterpart beyond the adapter can be replaced. The software architecture of simulated HRP-1S and the real one is shown in Fig.4. The controller API of V-HRP is not identical with that of Honda I/O board shown in Fig.3. The unification of a controller is realized by introducing the adapter whose API is the abstracted controller API mentioned above. The emulation adapter is layered on the controller API of V-HRP which reads the outputs of the sensor simulators and write the inputs of `DynamicsServer`, and the hardware adapter is put on the API of Honda I/O board of HRP-1S. The API of the adapters is identical shown as follows.

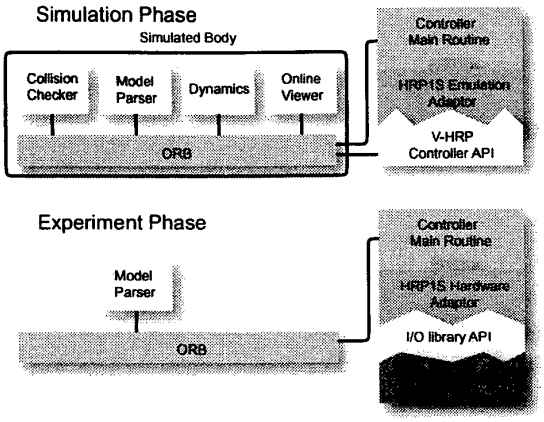

Figure 4: Software architecture of simulated and real humanoid robot HRP-1S

```
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);
};
```

3.2 Sychronization Mechanism

Though various speedup techniques for the dynamics simulation have been proposed to the present, but it is still difficult to compute the forward dynamics of robots with many degrees of the freedom including the visualization in realtime. Generally speaking, time goes by at a slower speed in the simulated world than in the real world. Besides, the speed is not constant in the virtual world due to the fluctuation of the required computation time.

Therefore, the sychronization mechanism of the controller must not be embedded in the main logic of the controller, but in the hardware adapter. Then the adapter can manage a change of the speed of time by a sychronization mechanism.

Note that the outputs of the controller can not always be updated in time in the real world. So it is possible that the controller fails to handle the robot even when it has succeed to move the robot in the virtual world.

3.3 Unified Controller makes The Development Efficient by Code Sharing

The controller and the dynamics simulator can share significant amount of codes. For example, collision detection is one of major building block 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 both the simulator and the controller. The parameters parser can also be shared. The forward kinematics computation of robots are common too. The unified controller makes the code sharing easier, and therefore the development of the controllers more efficient. Another good news of the code sharing is that the controllers can be more reliable since the building blocks borrowed from the simulator has been already examined intensively by the simulation.

However, there is a barrier to reuse the code in the controller. That is, the servers like `CollisionChecker` or `DynamicsServer` are implemented as CORBA servers as shown in Fig.2. Though realtime functions are included in the specifications of CORBA since version 2.4 and we can find the implementations of the functions, but the overhead of IIOP (Internet Inter-Orb Protocol) used in CORBA is not small enough for the controller which must update the outputs at a few milli-seconds. This overhead can be bypassed by the following architecture.

Let the servers like `CollisionChecker` or `DynamicsServer` consist of two layers. The lower layer is a normal library which is independent to 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(Object Request Broker) when reading a VRML file describing the pa-

rameters of robots, but they access non-CORBA interfaces when they control the robots in realtime. The internal configuration of the controller is shown in Fig.5.

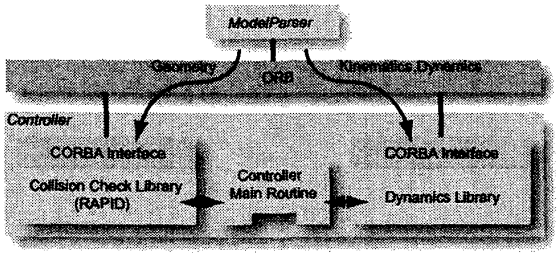

Figure 5: Internal configuration of the controller

3.4 Walking Experiment using Unified Controller

A walking stabilizing controller is developed on this software environment[10]. It controls body inclinations and ZMP using outputs of a gyrometer, an accelerometer and force/torque sensors. The upper graph of Fig.6 shows the ground reaction force measured by a force/torque sensor of the right leg while the humanoid is walking 4 steps in the virtual world, and the lower shows that in the real world. In both of these cases, the humanoid can walk stably and gotten reaction forces coincide sufficiently.

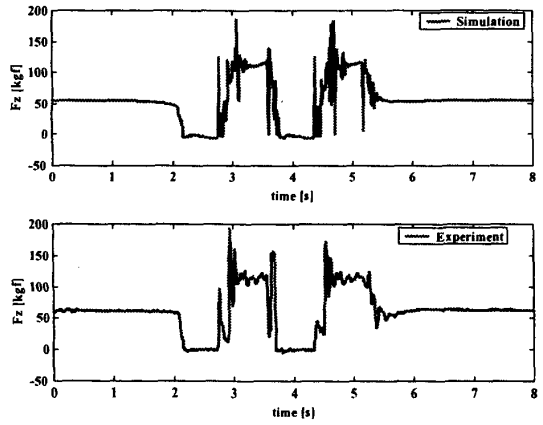

Figure 6: Ground reaction force while walking

Fig.7 shows body inclinations around a roll axis and a pitch axis while walking. These angles are estimated by the kalman filter using outputs of a gyrometer and an accelerometer.

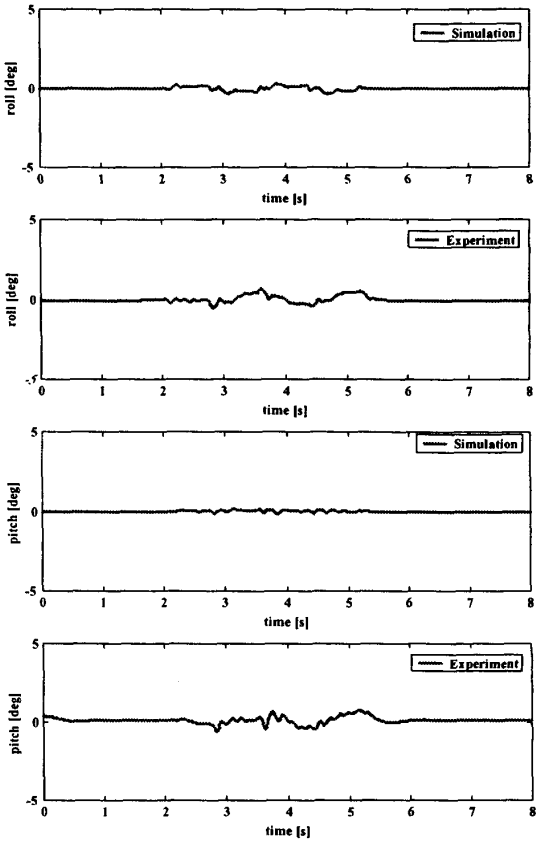

Figure 7: Body inclinations while walking

4 Realtime Collision Checker

A realtime collision checker is presented here as an example of shared codes between the simulator and the controller.

4.1 Self-Collision Check of The Humanoid Robot

When a humanoid robot is moving, it is desirable that the self-collision between the links of the robot is checked in realtime for enabling emergency stopping of the robot.

Let N be the number of the links of a humanoid robot. Then the number of pairs of the links is ${}_NC_2$,

and the collision detection must be executed for ${}_NC_2$ 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 ${}_NC_2 - (N-1)$ pairs. This number of the combination becomes 350 in the case of HRP-1S whose $N = 29$.

The average computation time for the collision detection is about 10 [ms], when HRP-1S takes many random posture. 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 its 20% can be assigned for the collision inspection. From the above experiment, about 35 pairs can be checked in 1 [ms].

The other extreme approach is inspecting self interference for all possible postures in advance, and storing the results in a table. But this approach does not work. Assume that the resolution of each joint is one degree and that the movable range of each joint is 90 [deg]. Then the total number of the collision detection is $90^{(N-1)}({}_NC_2 - (N-1))$. If the shape of a robot is symmetric, symmetric postures can be omitted. But the iteration number becomes 170×90^{28}, which is too big in practice. Besides, the lookup table should be too big as well.

4.2 Hybrid Collision Check Approach

We propose a hybrid approach of the realtime algorithm and the offline one.

Table 1 shows the number of colliding pairs between a link of the left arm and the left leg according to the joint angle of each joint. The joint angle increases from the left to the right in each row of the table, and the resolution of each joint is 11 degrees. Each cell shows the number of colliding pairs when moving joints while fixing the joint at certain joint angle. We can see that no pair of the links of the left arm and the left leg collides if the pitch joint of the shoulder is between -85 degree(minimum joint angle) and 14 for arbitrary combination of the remaining joint angles. The roll joint of the shoulder also has the safe range. Examples of the safe postures are shown in Fig.8, where the roll joint takes 61 degree.

The hybrid approach is summarized as follows.

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.

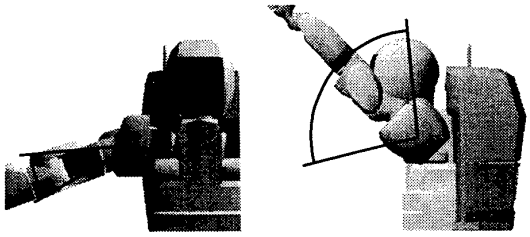

Figure 8: Safe posture for the collision between the arm and the leg in the same side

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

In the case of HRP-1S, we need not check the collisions between the arm and the leg in 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, 4 pairs between the main body and foot links, and 36 in total, which can be done in realtime. An example posture while walking is shown in Fig.9. An experimental result is shown in Fig.10. The required time is longest when the robot is standing, and decreases while walking.

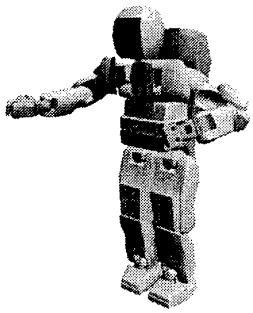

Figure 9: Safe posture for walking

Joint	min←Joint Angle→max																									
LLEG-HIP-Y	5	5	5	6	6																					
LLEG-HIP-R	5	5	5	5	6																					
LLEG-HIP-P	6	5	4	4	4	4	4	4	4																	
LLEG-KNEE-P	6	6	6	6	6	6	6	6	6	6	6	6	6	6												
LARM-SHOULDER-P	0	0	0	0	0	0	0	0	0	2	2	2	5	6	6	5	5	4	4	4	4					
LARM-SHOULDER-R	6	6	5	4	2	2	0	0	0																	
LARM-SHOULDER-Y	6	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	6	6	6	6	5	5	
LARM-ELBOW-P	2	2	3	4	6	6	6	6	6	6	6	6	6													
LARM-ELBOW-Y	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6			

Table 1: # of colliding pairs of the links

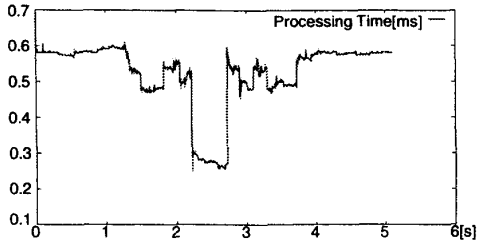

Figure 10: Computation time for the self-collision while walking

5 Conclusions

This paper presented a virtual humanoid robot platform on which we can develop the identical controller for a virtual humanoid robot and its real counterpart. The results can be summarized as follows.

- The unification of the controllers for the virtual and real robot has been realized by introducing the adapters for two robots respectively with the sychronization mechanism, and employing ART-Linux on which real-time processing is available at the user level.
- 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 realtime collision checker for humanoid robots has been developed as an example of the shared code.

Acknowledgments

This research was supported by the Humanoid Robotics Project of the Ministry of Economy, Trade and Industry, through the Manufacturing Science and Technology Center.

References

[1] Y.Nakamura, H.Hirukawa, K.Yamane, S.Kajita, K.Yokoi, K.Tanie, M.G.Fujie, A.Takanishi, K.Fujiwara, F.Kanehiro, T.Suehiro, N.Kita, Y.Kita, S.Hirai, F.Nagashima, Y.Murase, M.Inaba, and H.Inoue. V-HRP:Virtual Humanoid Robot Platform. In *Proc. of the First IEEE-RAS International Conference on Humanoid Robots*, 2000.

[2] Hirochika Inoue, Susumu Tachi, Kazuo Tanie, Kazuhito Yokoi, Shigeoki Hirai, Hirohisa Hirukawa, Kazuo Hirai, Shigeto Nakayama, Kazuya Sawada, Takashi Nishiyama, Osamu Miki, Toshiyuki Itoko, Hajimu Inaba, and Masako Sudo. HRP:Humanoid Robotics Project of MITI. In *Proc. of the First IEEE-RAS International Conference on Humanoid Robots*, 2000.

[3] Katsu Yamane and Yoshihiko Nakamura. Dynamics computation of structure-varying kinematic chains for motion synthesis of humanoid. In *Proc. of the 1999 IEEE International Conference on Robotics & Automation*, pp. 714–721, 1999.

[4] http://www.omg.org. Object Management Group.

[5] S. Gottschalk, M. C. Lin, and D. Manocha. OBB-Tree:A Hierarchical Structure for Rapid Interference Detection. In *Proc. of ACM Siggraph '96*, 1996.

[6] http://www.h-anim.org/. HUMANOID ANIMATION WORKING GROUP.

[7] Youichi Ishiwata and Toshihiro Matsui. Development of Linux which has Advanced Real-Time Processing Function. In *Proc. 16th Annual Conference of Robotics Society of Japan*, pp. 355–356, 1998.

[8] http://luz.cs.nmt.edu/~rtlinux. *RT-Linux.* V. Yodaiken and M.Barabanov.

[9] Fumio Kanehiro, Masayuki Inaba, Hirochika Inoue, Hirohisa Hirukawa, and Shigeoki Hirai. Developmental Software Environment that is applicable to Small-size Humanoids and Life-size Humanoids. In *Proc. of the 2001 IEEE International Conference on Robotics & Automation*, pp. 4084–4089, 2001.

[10] Kazuhito Yokoi, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara Shuji Kajita, and Hirohisa Hirukawa. A Honda Humanoid Robot Controlled by AIST Software. In *Proc. of the IEEE-RAS International Conference on Humanoid Robots*, 2001.