2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

Realization of Dynamics Simulator Embedded Robot Brain for Humanoid Robots

Takashi Ogura, Kei Okada and Masayuki Inaba

Abstract— **This paper proposes the new robot programming environment in which robot motion programming environment and dynamics simulator are integrated. This allows robot motion programs to include simulation descriptions. Additionally, a new implementation of simulation that is composed by simulation modules is presented, on the other hand, conventional simulators are monolithic and implanted every function. This makes it difficult to add new simulation functions such as new sensors on a simulator by its users. In the new method, the users of the environment can add new modules easily. The simulation function of this system is evaluated by showing new robot motion simulations like brooming, seesaw and so on. The experiment that shows how the simulation embedded brain changes the motion planning of block moving problem is illustrated in the end this paper.**

I. INTRODUCTION

This paper proposes "Dynamics Simulator Embedded Robot Brain" and show how to realize it. This robot brain has dynamics simulator inside and it can make motion planning, learning or prediction using the simulation environment. The left figure of **Fig.**1 shows usual relation of humanoid robot brain and dynamics simulator such as FAST[1] or OpenHRP[2]. The brain is connected to the real robot or the virtual robot. The virtual robot is just substitution of the real robot. In such environments, it is difficult to control virtual world from the brain, and only can do batch simulations. The brain can not handle the dynamics models in the simulator. Therefore the environment is not enough for the brain to learn or predict future. The new brain has simulator inside, and uses it for prediction and makes motion planning or controlling of motion. The right hand of the **Fig.**1 shows that. This simulation embedded brain has the three merits mainly not only for the brain itself but the users.

- Sensor-based programming in only one environment.
- The brain can control the virtual world directly.
- Interactive simulation.

At first, if the brain does not have simulators outside, it can learn by try and error in simulator using only this system.

Tryy and error learning is done in simulators so offten[3], [4], and the brains usually use simulator for learning using ones outside of them. Then the brain must connect the virtual world and synchronize the two environments. This is more troublesome for the brain and the users. The second benefit is adjustment of virtual world. If the brain build the virtual world from recognition, the brain should have full control of the simulator. The objects in the simulator have many parameters such as friction, spring in collision and so on. The third merit is that the simulation becomes interactive. If the brain controls the simulator, the simulator must controlled interactively. Interactive simulation is also useful for the human.

This paper proposed "Dynamics Simulator Embedded Robot Brain" for humanoid robots at first. In the next section, we show the functions the brain should have. Then we discuss how to implement the environments, and show the new implementation method. We describe EusDyna, which is implemented by the method, and show some simulation results of humanoid robots. The experiment that shows how the simulation embedded brain changes the motion planning of block moving problem is illustrated in the end this paper.

This work was not supported by any organization

T. Ogura (JSPS Research Fellow), K. Okada and M. Inaba are with Department of Mechano-Informatics, Graduate School of Information Science and Technology, The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan {ogura, k-okada, inaba}@jsk.t.u-tokyo.ac.jp

II. THE FUNCTIONS FOR SIMULATION EMBEDDED ROBOT BRAIN

For realizing Simulator Embedded Robot Brain, at first the brain and dynamics simulator must be integrated. One of integrated robot programming environment is OpenHRP. OpenHRP's features are realization of distributed processing using CORBA, and to be able to use the same binaries for simulators and real robots. However, this is for human not for robot brain. When robot brain uses simulator, more three features are required such as,

- selectivity in space
- selectivity in time
- interactive control

In the following, it explains these.

A. *Selectivity in Space of Simulation Target*

In conventional simulators, all objects and agents in the simulator are simulated. If there are many objects, the calculation amount becomes huge. Robot brain does not need the result of all objects, but the objects payed attention to by the robot. Then we call the function to select for simulation "selectivity in space". The robot is focusing on the cylinder in **Fig.**2, then it only needs the simulation result of the cylinder.

B. *Selectivity in Time of Simulation Target*

The brain must have "selectivity in time" as same as "selectivity in space." The brain does not always need the simulator. If it always simulates, most of the results are wastes. It is important to be able to select target, start simulation and stop it in any time, and back to the past.

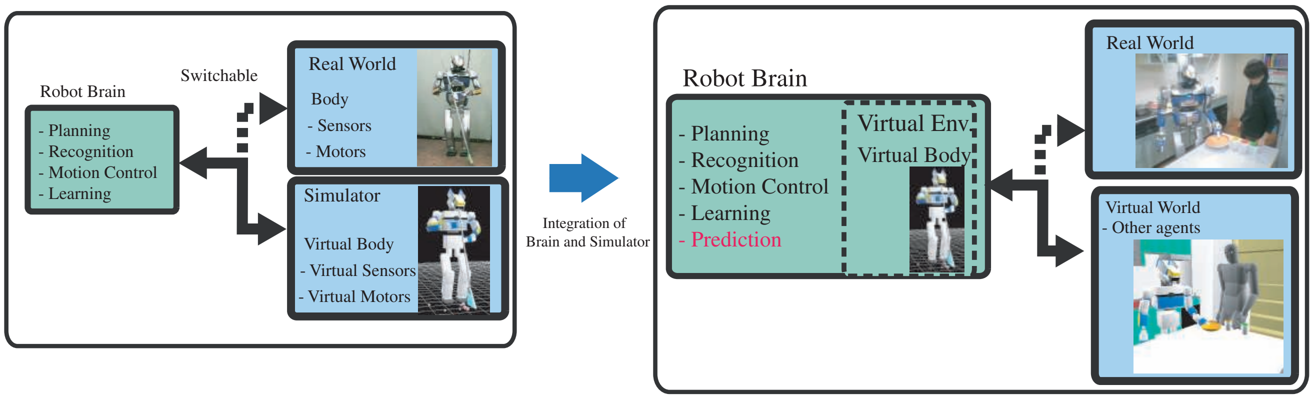

Fig. 1. Conventional brain and simulator, new simulator integrated brain.

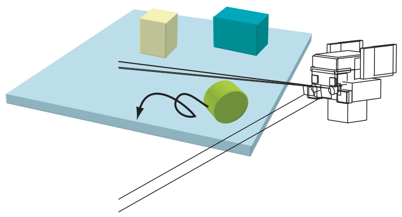

Fig. 2. Simulate the objects paid attention to by the robot

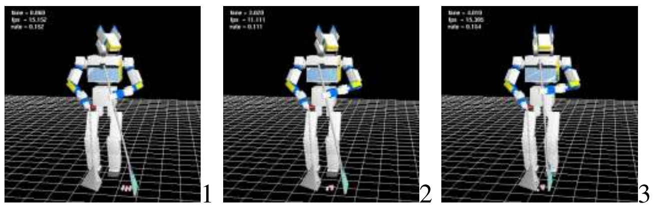

Fig. 4. Garbage collection motion using dustpan and broom (configured parameters)

C. *Interactive Control of Simulation*

In the complicated simulations such as humanoid robots use tools of human, there are many simulation parameters like friction or hardness. The brain must be able to control simulation world interactively. **Fig.**3 shows the simulation that the humanoid robot cleans using a broom. At first, the hardness of the broom head is too hard to collect the garbages, and this is not natural. If the coefficient of bounce is changed by the brain, the brain can simulate the sweep well (**Fig.**4). These motion program are the same one, but only the coefficient is different. Not only adjustment of parameters, but simulation methods or simulation speed should be controlled by the brain.

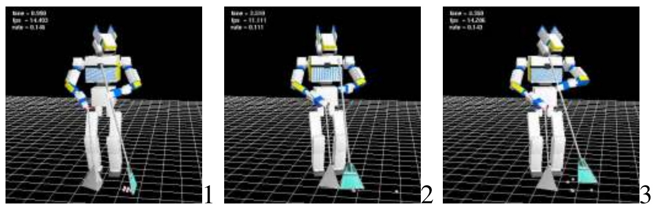

Fig. 3. Garbage collection motion using dustpan and broom (not configured parameters)

III. IMPLEMENTATION OF SIMULATOR EMBEDDED ROBOT BRAIN

A. *Implementation of Simulator by Micro-kernel method*

In this section micro kernel method, which is one of implementation method of simulator, is described. In simulator programs, integration, collision, servo simulation, and simulation of many kinds of sensors are done for one loop, and repeat it. Left side of **Fig.**5 shows this flow. This structure is normally static. If you want to add functions or change, you must recompile the simulation programs. This is not easy for the users, and can't be done dynamically. We call this conventional implementation method of simulator, monolithic kernel method. The core of simulator is monolithic and can not be divide. On the other hand, this paper proposes micro kernel method. The system prepares only the framework and minimum functions, and users can add or delete functions dynamically. Right side of **Fig.**5 shows this method. The functions such as servo simulation are implemented in each modules, and the system calls this in turn. The each module has only one function. The robot brain can add any modules if new simulation method is required. This addition of modules can be done dynamically while simulation is running. All the modules have common interface between the modules and the system, and each module knows which object is target of simulation.

In addition, it is considered that the simulator should be able to deal the phenomena that can not be simulated physically, such as switching of light or mechanical systems. Because these not physical simulation function is very many, these can not be implanted in advance. In micro-kernel method simulator, these functions are added when it is needed. **Fig.**6 shows "breakout" game in simulator. In the game, the blocks collided with the ball disappear. This is not physical function. If you use monolithic simulator, you must reconstruct the simulator at all when you add the function.

However if you use micro-kernel method simulator, you can add the function very easily.

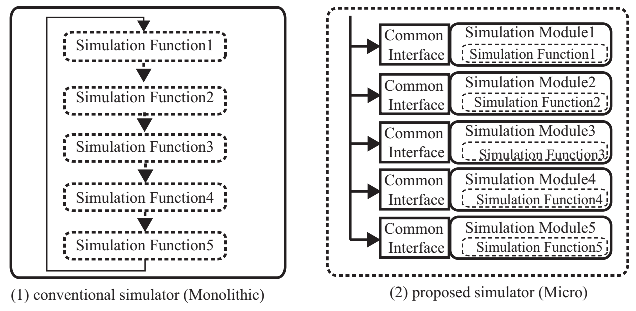

Fig. 5. Implements of conventional simulator (Monolithic) and proposed method(micro-kernel)

Fig. 6. 3D Block Game by simulation

B. Example of Implementation: EusDyna

We developed EusDyna, which realizes "Robot Brain with Simulation Prediction Function." EusDyna is based on EusLisp[5]. EusLisp is one of Lisp implementation, and it has geometric modeling and multi-thread functions. The simulation model is also modeled by the modeling function of EusLisp. We have developed many motion programs using EusLisp, then EusDyna can utilize the resources. We use EusLisp for planning in symbolic and static geometric environment normally. EusDyna expand the planning for dynamics geometric world.

Simulator part of EusDyna is implemented by the micro-kernel method as above. **Fig.**7 shows the programs in the interpreter and the thread. In the thread, modules like **Table** I are called in turn in one step, this makes simulation. This thread loop can be controlled by interpreter, and you can execute only one step too. The core physical module uses external libraries. Because this core is one of modules, it can be exchanged. There are four core modules that uses ODE[6], Math Engine of Vortex, PhysX[7] that can use PPU(Physics Processing Unit) and only EusLisp. EusDyna can use one of these core modules and any other modules. The user can use these libraries without consciousness of the difference because these libraries are encapsulated. Management of modules and viewer are implanted on simulation kernel. In EusDyna, modules in **Table** I realize simulation.

C. API of EusDyna and Example Codes

Main APIs of EusDyna are shown in **Table** II. Characteristic ones are "d-tick" for proceed simulation only one

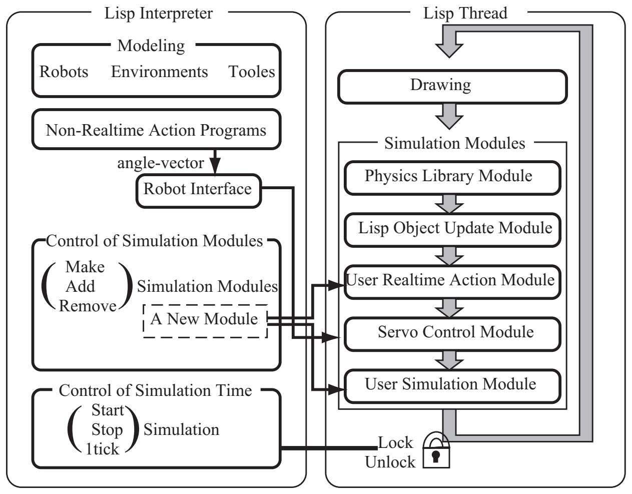

Fig. 7. Configuration of EusDyna: Users make models of robots or environments, motion programming and control simulation from interpreter. The thread draws the results of simulation and executes each modules. When users control simulation, the system uses mutex lock.

TABLE I
EXAMPLES OF SIMULATION MODULES

Module Name	Function
Dynamics	Collision and Integration. Simulation Core
Servo	Control of Servo Motors
Force Sensor	Simulate Force Sensors
Acceleration Sensor	Simulate Acceleration Sensors
Gyro Sensor	Simulate Gyro Sensors
ZMP Sensor	Simulate ZMP Sensors

step, "d-make" for selecting the target for simulation or "d-no-collision" for setting collision free pairs. The robot brain uses these APIs for prediction or learning.

A sample code is shown in **Fig.**8. At first, it makes two cubes (make-cube) and sets colors and weights. :locate method sets the position of the cubes. The rest of d-init, d-make, d-sart, objects are EusLisp code. These functions do initialization, select simulation targets, start simulation and display. This sample code runs the simulation shown in **Fig.**9 without any model files or other simulators.

In addition, simulation of game, buggy, robot arm or humanoid robot are written in 100–200 lines like **Fig.**10. Simulation speed of EusDyna is depend on mainly the libraries. If it uses ODE, the system takes $740msec$ for $1sec$ of simulation of humanoid like in **Fig.**4 (CPU: Pentium D 3.45GHz). This shows that the robot can use the simulator in real time.

IV. SIMULATION OF HUMANOID ROBOT

Simulator integrated robot brain system made new simulations of humanoid robot possible using selectivity of Space and time or interactivity of simulation. In this section simulations are illustrated.

Fig.11 is a basic simulation of humanoid robot. The humanoid robot walk to a ball and kick it. Such simulations are major because these do not need the features described in this paper. More complicated simulations are shown here.

TABLE II

EXAMPLES OF EUSDYNA API

API Name	Function
d-init	initialize simulation
d-end	simulation termination
d-tick	proceed simulation 1 step
d-start	start simulation thread
d-backtick	back simulation 1 step
d-make objs	make "objs" simulation targets
d-make-joint b1 b2	make a joint between b1 and b2
d-no-collision objs	ignore collision among "objs"
objects objs	display "objs" on viewer

```
1   (setq aa (make-cube 100 100 100))
2   (setf (get aa :face-color) :blue)
3   (setf (get aa :weight) 50)
4   (setq bb (make-cube 200 200 150))
5   (setf (get bb :face-color) :yellow)
6   (setf (get bb :weight) 60)
7   (send aa :locate #f(50 50 600))
8   (send bb :locate #f(0 0 800))
9   (objects (list aa bb)) ;; display
10  (d-init) ;; initialization
11  (d-make (list aa bb)) ;; select
12  (d-start) ;; start simulation
```

Fig. 8. An Example of EusDyna's code

The simulation which uses selectivity of space is shown in **Fig.**12. The humanoid robot hold a dish and take it on a tray, and carry it using the tray. Because the robot brain selects only dishes, tray and board for simulation target, the simulation can be run faster in such complicated situation.

Fig.13 shows the simulation results of sweeping garbages with a broom. This simulation requires selectivity of time. **Fig.**14 shows the main codes of this simulation and motion. Geometric models are made in the first and second line. This models are static one, and used for making poses or motion planning. The lines start with "send" is calling methods of the robot model. In the methods, if the robot is selected for simulation, do the simulation, if not selected only animation. In this codes reset motion and grasp the broom model, then initialize simulation and select robot, broom and garbages for simulation. Simulation is done between "d-init" and "d-end." After confirm the motion the robot brain can do the same motion on the real body. The motion on the real body is shown in **Fig.**15.

If the brain needs moving floor for simulation, it can add moving floor module easily. The sample code for making and adding the module is **Fig.**14. Line 1 to 7 defines class and make instance of it in line 9. Linke 10 adds the instance to the system. The micro-kernel method enables this. The simulation results is shown in **Fig.**17. The floor moves too fast for the humanoid robot to keep standing.

Next example is reinforcement learning using this system. **Fig.**18 shows the real robots environment of "swing" and "see-saw". The brain can do reinforcement learning without any outer simulators and use results for the real body. **Fig.**19 shows that the humanoid robot sit on swing, and moves the legs for acquiring acceleration. EusDyna can deal not only one robot. **Fig.**20 is another sensor based motion example. Two humanoid robots ride on see-saw and kick the ground when the foot touched the ground.

Fig.21 shows simulation with visual processing. The system can simulate robot vision functions as a module. The robot pull a drawer and finding an object by the stereo camera on the head.

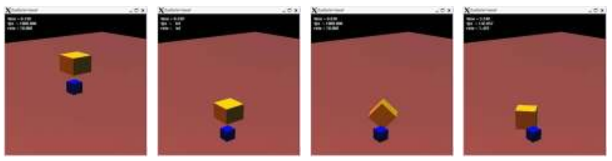
Fig. 9. Simulation results by the example code

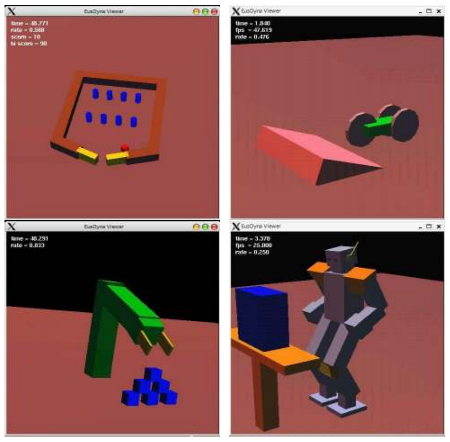
Fig. 10. Examples of simulation: Block game, buggy, robot arm and humanoid robot

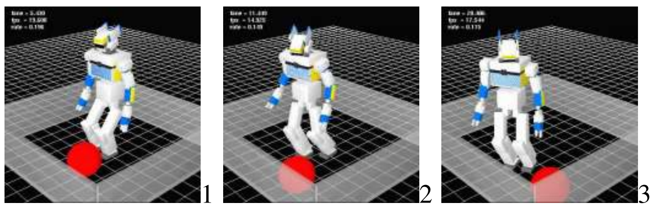

Fig. 11. Walking toward a ball and kicking it

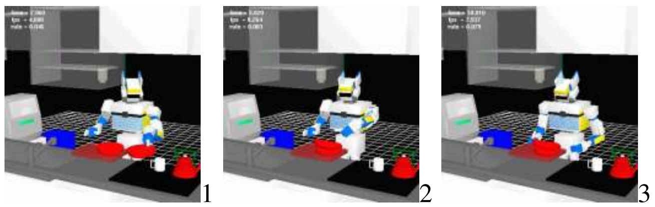

Fig. 12. Tidying up motion (stack the dishes and carry them using a tray) in a kitchen environment

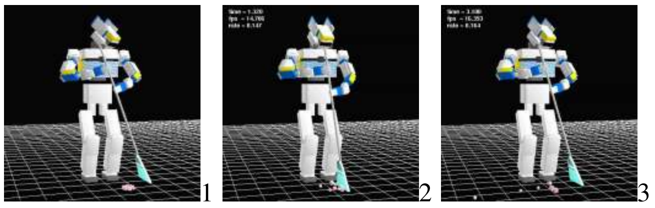
Fig. 13. Brooming simulation

```
(setq *bd* (create-broom-dust-model))
(setq *robot* (create-robot-model))
(send *robot* :reset-pose)   ; reset motoin
(send *robot* :hold-broom)
(d-init)                     ; initialization
(d-make (list *robot* *bd*)) ; select target
(dotimes (i 3)
  (send *robot* :swing-broom)); sweeping
(d-end)
(send *robot* :reset-pose)   ; reset motion
```

Fig. 14. Motion programming code of brooming simulation

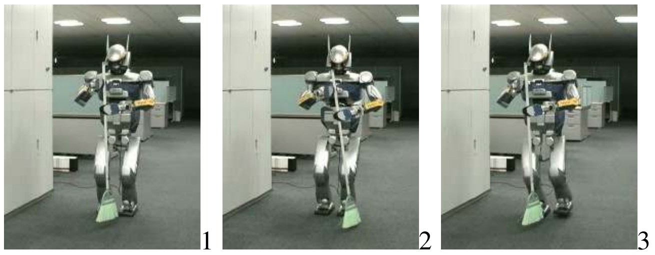
Fig. 15. Scene of brooming by Humanoid Robot HRP-2

```
(defclass earth-quake-module
  :super eusdyna-module
  :slots ())
(defmethod earth-quake-module
  (:control (obj)
    (send dj :angle (* width (sin x)) (* 1000 dt)))
    (incf x speed)))
(setq *dworld* (d-init))
(setq *em* (instance earthquake-module :init))
(send *dworld* :add-module *em*)
(send *robot* :stabilize-start) ;; control brain
```

Fig. 16. Motion programming code of moving floor

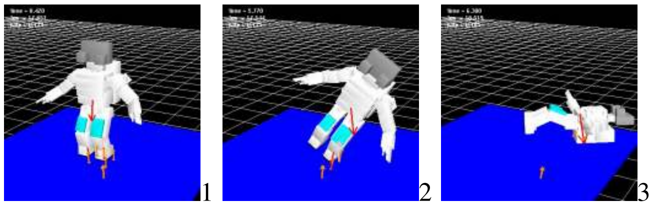
Fig. 17. Simulation result of moving floor

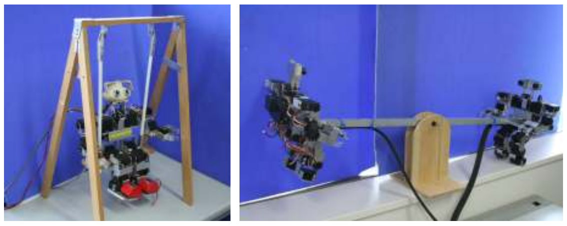
Fig. 18. Swing and see-saw environment in real world

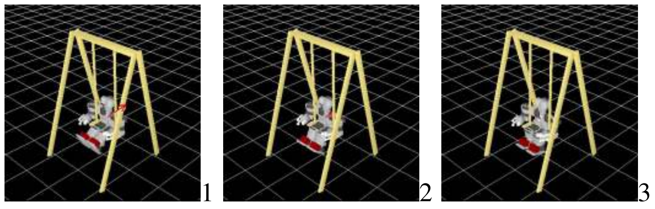
Fig. 19. Swing simulation: Reinforcement learning

V. MOTION PLANNING USING DYNAMICS SIMULATOR

A. *Overview of Motion Planning*

This section illustrates a motion planning using dynamics simulator. Integration of brain and dynamics simulator enables that recognition, making model, simulation, and generate motion plan using the simulation results. **Fig.**22 shows the flow of this.

B. *Block Moving Task using Dynamics Simulator*

We show block moving task for an example. There are three blocks, and they are stacked. The task is that the under block should be moved to left side position without any blocks over. Normal robot brains generate the plans that move blocks one by one in order very carefully. But this is not like human and too polite. The wiser agents like human will act more dynamically.

At first the robot acquires the block color, size and position by stereo vision. The left pictures of **Fig.**23 are visual images of the robot, and the rectangle regions that are detected as the blocks. The right sides are the models in the robot brain. The upper images are the initial scene. The models are created in this scene. The blocks are tracked and the results are lower images. The robot brain with simulator can use the block models for simulation directly.

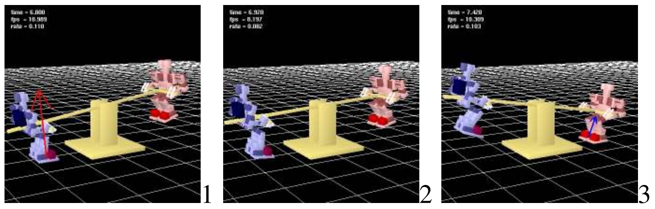
Fig. 20. See-saw simulation by two humanoid robots: these robots have force sensors on their foots, and kick the ground using the sensor values.

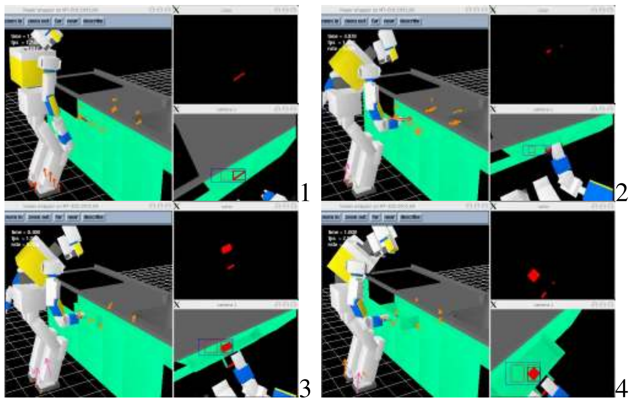

Fig. 21. Integration with vision processing: Pull the knob recognizing the position by stereo vision (The right upper picture is result of color extraction, Left lower one is result of stereo vision processing in each pictures.)

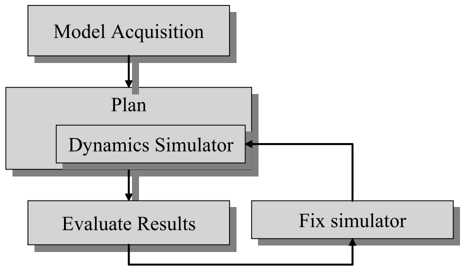

Fig. 22. Flow of motion planning using dynamics simulator

The planner selects the block for moving and selects moving direction for one action. In this case, A* search algorithm is used and generates the orbit of blocks. The heuristics is the distance to the target position and number of blocks on the target block. If any block falls down on the ground the search cost is infinity. The planner generates robot whole body motion from the orbit of the blocks, using inverse kinematics and stabilize the pose.

A simulation module is used for moving the blocks. This enables the motion of blocks that is not natural without simulation robot motion. The motion and movement of blocks are shown in **Fig.**24.

VI. CONCLUSIONS

This paper proposed Robot Brain integrated with dynamics simulator. Therefore we implemented EusDyna and described it. Then some simulation results of humanoid robot with tools of human, and sample codes are shown. This system can start and stop simulation in any time, and can select simulation targets, then brooming by a humanoid robot and simulation in complicated situations such as kitchen space. Additionally, a new implementation method of simulator is shown. The mechanism makes it possible for the brain to add or delete simulation functions dynamically. The block stacking problem experiment shows that the new brain can generate dynamic motions which may be looked rough, but like humans do. It needs the functions that absorb the difference between the predicted situation and the real, and customize the simulator on site. The system shown in this paper can change the robot's behaviors very much.

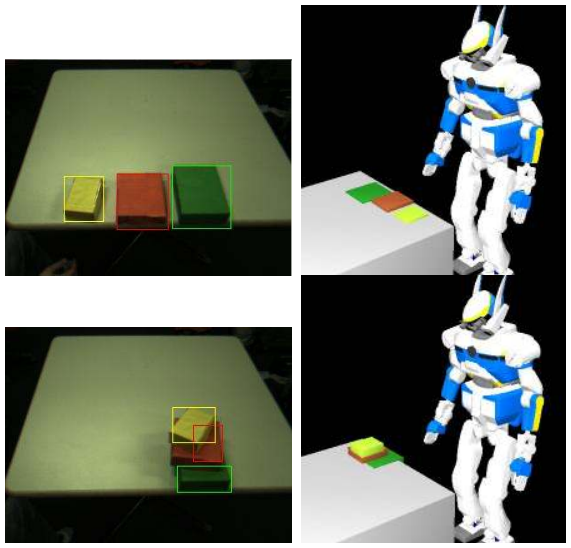

Fig. 23. Scene of acquisition models by vision (Left: visual image and detected regions. Right: acquired model of blocks in the robot brain.)

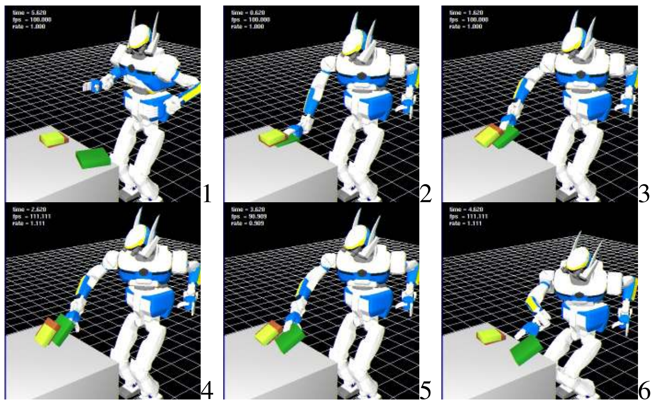

Fig. 24. Results of motion plan using dynamics simulator

REFERENCES

[1] K. Okada, Y.Kino, F. Kanehiro, Y. Kuniyoshi, M. Inaba, and H. Inoue. Rapid development system for humanoid vision-based behaviors with real-virtual common interface. In *Proceedings of the 2002 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS'02)*, pp. 2515–2520, 2002.
[2] Fumio KANEHIRO, Kiyoshi FUJIWARA, Shuuji KAJITA, Kazuhito YOKOI, Kenji KANEKO, Hirohisa HIRUKAWA, Yoshihiko NAKAMURA, and Katsu YAMANE. Open architecture humanoid robotics platform. In *Proceedings of the 2002 IEEE Intl. Conference on Robots and Automation (ICRA'02)*, pp. 24–30, 2002.
[3] K. Sims. "Evolving Virtual Creatures" Computer Graphics. In *Proceedings of Siggraph*, pp. 15–22, July 1994.
[4] J. Bongard and R. Pfeifer. Evolving complete agents using artificial ontogeny, 2003.
[5] T. Matsui. Multithread object-oriented language euslisp for parallel and asynchronous programming in robotics. In *In Workshop on Concurrent Object-based Systems, IEEE 6th Symposium on Parallel and Distributed Processing*, 1994.
[6] Open Dynamics Engine ODE. `http://ode.org`.
[7] PhysX. `http://www.ageia.com`.