An Open Robot Simulator Environment

Toshiyuki Ishimura, Takeshi Kato, Kentaro Oda, and Takeshi Ohashi

Dept. of Artificial Intelligence, Kyushu Institute of Technology
isshi@mickey.ai.kyutech.ac.jp

Abstract. At present, various kinds of robots such as AIBO, ASIMO and etc, are available in public. However, the development of robots is still having some difficulties since of their complexity, continual changes of environments, limitation of resources and etc. To overcome this problem, robot developers often use the simulator that allows to program and test robots' program effectively under ideal environmental conditions where specified various conditions can easily be reproduced. It is still difficult to realize the simulator regardless of its usefulness, because the cost of simulator implementation seems the unexpected cost in the development of robots. As a result, it is need to realize the open robot simulation environment in which any kind of robots can be simulated. This paper focuses on vision-based robot simulation environment and describes a method to construct it. Finally, we implemented a simulator for Robocup Sony 4-Legged League by using this method.

1 Introduction

Nowadays various robots are available in public such as AIBO, ASIMO and etc. However, the development of robots is still having some difficulties because of their complexity, continual changes of environments, limitation of resources and etc. Considering environment changes in vision-based robot, for instance, changes of the lighting condition in a specific environment will affect robot's behavior seriously. To understand the problems of the robot's strategies in the real environment, it needs to check robots strategies in exactly the same environment conditions. Because of in each testing time, sensory values such as camera images and the effectors will change. There are two types of robot simulators in order to solve this problem. One aims to simulate the robot mechanical behavior with accurate robot model data and the other aims to simulate the vision of the robot in order to test robot strategy. The simulator allows developers to program and test robots' program effectively in the ideal environment where specified conditions can easily be reproduced. It is still difficult to realize the simulator regardless of its usefulness since the cost of simulator implementation can be an unexpected cost in robot development. Can overcome this problem by realizing the open robot simulation environment as it can simulate any type of robots.

This paper focuses on vision-based robot simulation environment and describes a method to construct it. The next section describes concept of open robot simulator. Section 3 shows architecture of this method. Section 4 presents an implementation of the environment. Section 5 shows evaluation of the simulator for Robocup Sony 4-Legged League. Finally, section 4 concludes with a discussion of our method.

D. Polani et al. (Eds.): RoboCup 2003, LNAI 3020, pp. 621–627, 2004.

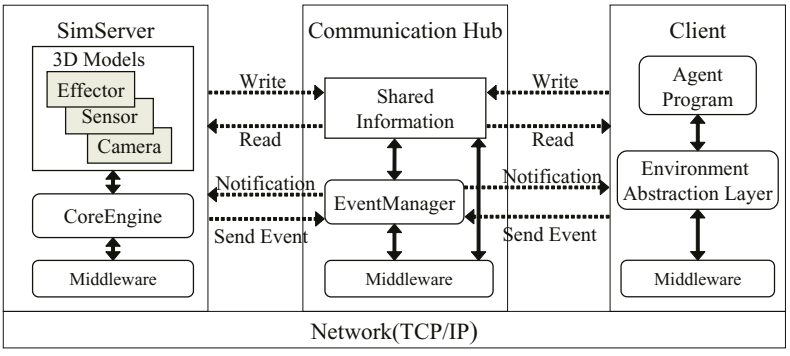

Fig. 1. The Simulation Environment Architecture

2 System Design

The purpose of the method is to improve the development efficiency of the vision-based robots' strategies. To achieve this, we implemented an open robot simulation environment so that different robot developers can introduce different robots to the environment in order to realize the functions of the simulator. The overall policy of the method is as follows.

- Openness is the main focus in the system design stage. With high openness, developers can manipulate the simulation environment easily to treat various kinds of robots. It also allows customizing simulation environment according to the real environment to introduce new robots, new planning strategy, new color modules, new collision detection module, new image processing module, etc
- Reproducibility - Real robot always changes the behavior according to the environmental condition changes such as lighting condition. Since testing of robot strategy consumes number of development cycle consists of coding and debugging, the simulator must support to reproduce a certain situation.
- Distribution in a Network - Since individual robots exist distributed in a physical space, the environment must support distribution in a network.
- Minimize the Requirements - When migrating between real robot and the simulator, it needs to reduce the cost of migration. In our method it is possible, since the robot's strategy program having less modification and its easiness to modify

3 Architecture

Fig 1 illustrates the system architecture, which adopts client/server model and introduces two servers called the simulation server (*SimServer* for short) and the communication hub (*ComHub* for short). In this model, there are multiple heterogeneous robot agents, which are implemented in different program languages and can participate to this environment because clients connect to the server via the network.

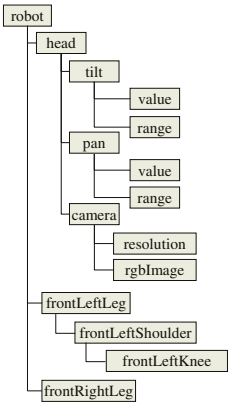

Fig. 2. A Tree Representation Example of SimServer's Objects

```
<?xml version="1.0" encoding="UTF-8" ?>
<Environment version="0.1" xmlns="http://www.asura.ac/xethereal">
  <Leaf name="/Robo0/robot/head/tilt/value" type="double" scalar="true">
    <i>32.5</i>
  </Leaf>
  <Leaf name="/Robo0/robot/head/pan/value" type="double" scalar="true">
    <i>11.4</i>
  </Leaf>
  <Leaf name="/Robo0/robot/leftLeg" type="double" scalar="false" length="3">
    <i>1.5</i> <i>131.4</i> <i>231.1</i>
  </Leaf>
  ....
</Environment>
```

Fig. 3. An Example XML Output of Environment Information

3.1 Simlation Server

The *SimServer* manages all the objects in the simulated environment and synthesizes virtual camera images and simulates the robots' effector. To reduce the cost of realization of the simulator, the system provides a class library of fundamental component which are implemented with opened interface in order to operate easily. The following list show the main part of the class library.

Robot holds cameras, effectors and its position and orientation.

Camera holds parameters such as view angle, resolution and so on, synthesized images by the Core Engine.

Effector keeps current value and the range of values. It also can hold any sub-effector and camera as children.

This library allows users (i.e. robot developers) to construct the virtual robot by few steps; combination of any object according to real robot, customization of object parameter such as camera resolution and the range of effectors, and corresponding to existing 3D shape model data of the robot. The *SimServer* manages all the objects in tree structure and provides name space according to that structure so as to developer can get information user friendly. The robot's strategy program and developers can access all the objects and its all attribute information by that name. Fig 2 shows an example of tree structure of the Sony ERS-210.

Any object in the *SimServer* can be appended and removed in runtime to configure the simulation environment dynamically. For example, when a new robot participates in the simulator, the *SimServer* create new objects according to robot template on demand.

3.2 Communication Hub

The communication hub (*ComHub* for short) holds whole environmental information including from each robot's estimated position in the soccer field to camera images generated through the *SimServer* .

Snapshot. The hub also can take a snapshot of its contents. The snapshot is XML formatted so that programmers can modify and export. (Fig 3 gives an example of the XML output result.)

Dynamic Connection/Disconnection of Clients. This is one reason why we introduce the hub. In development cycle, we frequently kill/run the clients. Using this hub, the system can continue the execution regardless of disconnection because the last state of a robot is still hold even if it was crashed.

Dual Communication. We provide two communication mechanisms - one is synchronous operations: `read()` and `write()` operations to the tree. These operations are fairly simple. To read value on the path name of /robot/head/camera/rgbImage is `read("/robot/head/camera/rgbImage")`. It returns a byte array container. To gain the performance, bulky versions of these operations are provided. The other is an asynchronous event mechanism which supports send and listen event operations. All events must be sent with a string label to address the contents. The event receiver can set a filter to receive interested events and reduce bandwidth by specifying a regular expression. Only events matches the regular expression can be received.

A communication pattern may be used in common is like that 1) a client receives an update event, and 2) issues `read()` operations to get its interested nodes, 3) send an finish read event. 4) After receiving the finish read event, a client who wants to write data start to write the nodes through `write()` operation. After that, 5) it sends an update event. The tree held in the communication hub works as a shared memory. Asynchronous event mechanism can be used as a synchronization mechanism between data producer and consumer.

3.3 Functionality

The following functionalities enhance the simulation environment: plugin user module, a script language, persistence of simulation environment state, communication among the agents and visualization of view frustum.

First, developers can insert fragment of program as plug-in module to the virtual robot in order to customize its behavior. The environment provides interfaces for plug-in modules such as image transformation plug-in, effector plug-in and constraints plug-in. By using this functionality, user can realize strange camera image, reduction of effector speed, etc without time consuming task. For instance, if a simulator should provide the YUV image, the robot developer only prepare a plug-in which transform from RGB to YUV color space. In fact, our team implemented the plug-in described a few codes, since Sony ERS-210 generates YUV camera image. Since every plug-in module is as the *SimServer* 's object, they can be appended / removed and enabled / disabled by scripting at runtime.

Second, to provide interaction between developers and the *SimServer* , a script language likes the S-Expression is introduced. The following enumeration shows typical features of the scripting:

- Getting and setting values of all the objects in the environment
- Loading new objects and plugin at runtime

```
(let
  (set /Ball/loc/x
    (plus /ribo0/loc/x
      (/ (dist /robo0 /robo1) 2)
    )
  )
  (set /Ball/loc/y
    (plus /ribo0/loc/y
      (/ (dist /robo0 /robo1) 2)
    )
  )
)
```

Fig. 4. An Example of script language

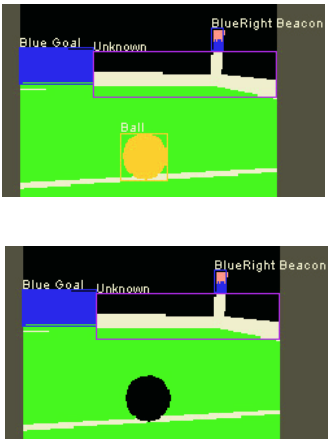

Fig. 5. An Example of image transformation plug-in

– Removing all the objects and plugins at runtime
– Configuration of system setting such lighting condition, frame rate of the simulation
– Easy to introduce new commands

This allows us to test the robot's strategy in the exactly the same environment because the simulator can reproduce it repeatedly. Fig 4 shows an example code to place a ball between two robots (named `robo0` and `robo1`).

Third, the *ComHub* can export its snapshot to a file as described the above. By using this feature, the system can reproduce a certain situation according to the snapshot. This functionality increases reproducibility.

Forth, by using the *ComHub* 's feature that holds whole environmental information as shared information, the system provides communication among the each client. The *ComHub* allows the *SimServer* and every client to put different information to it. When putting information, the *ComHub* notifies this update to the *SimServer* and each client.

Finally, when using the active camera, enhancement of camera motions is important to recognize the virtual environment. The simulator provides functionality to visualize the view frustum that each of the cameras is now seeing. This visualization is useful in order to adjust and tune-up camera motion.

4 Implementation

Initially, we implemented the *SimServer* and the *ComHub* by using Java and Java3D [2]. The *SimServer* is shown in Fig 6 and it consists of four components; global view, command line panel, the tree view, local camera views.

On the global view, user can change his own view by mouse operations. The command line panel allows us to interact with the simulator by the script language described in the above. The tree view shows information about all the objects in the simulation environment.

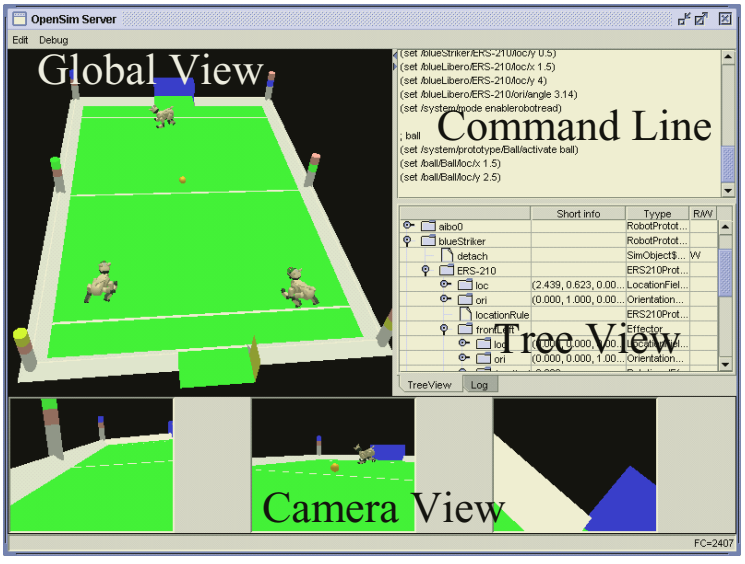

Fig. 6. The overview of the *SimServer*

Second, to evaluate the method, we implemented a simulator for the Robocup Sony 4-Leggued League on the environment. Then we succeed in migration from real robot's strategy program [1] to the simulator with minor modification.

5 Evaluation

We measured the frame rate of the simulator on the PC Intel Pentium 4, 2.8GHz, 1024MB Ram with ATI RADEON 9700 Pro video card and three robot agents connected to the simulator via 100MB LAN. As a result, each client worked at almost 6.2 fps (frame/seconds).

At this time, the main differences arise in the simulator environment against the real field one are that lack of a robot physical model, accurate sensory values, ideal effectors (no slippery walking), no collision to the ball, robots and the field boundaries, noiseless synthesized local vision images, relatively rich computation resources rather than ERS-210, lesser inter-robot communication latency. Most of the factors would be dealt with the introduction of new plug-in modules (like a noisy camera filter). But depending on applications, these differences could break precision of a target simulation and make the results unusable. However, as far as we have tested the simulator from 1 years ago, in our application practice shows its strong effectiveness. The following is the some practical examples.

1. Multi-agents coordination programming: The simulator can accommodate multiple-agents with its inter-agent communication facility. A programmer checked an inter-agent information sharing mechanism, ball occlusion test (Fig 5), a lot of team formation strategy. Through this process, the programmer successfully created a coordination algorithm which applied in a real game.

2. Vision module debugging: A programmer discovered a serious vision code bug which appears very occasional. With the simulator, debugging can be done effectively because of rich development environment.
3. Robustness test: A lot of the audience wears colorful clothes which can easily confuse the robots vision system. To minimize the influence, we introduced a filter which eliminates too high markers. The simulator can create an ideal bad environment in a fraction of time.
4. Education: For aibo programming beginners, we provided the simulation environment as a primary test bed. This accelerates their learning curve. The simulator never hurts real robots so the learner can try new things freely.

6 Conclusion

On the vision-based robot simulator, it is important to reduce the cost of robot strategy programming. To achieve this, we proposed the open robot simulation environment to accommodate any kinds of robots by using standard Java, distribution in a network with TCP/IP, minimized the requirements through simple communication operations, providing rich information through the tree structured external representation, openness with communication hub and plug-in facility in the simulator and rich debugging facility.

In the four-legged league in RoboCup 2002, our system had not been implemented. Now we have a good fundamental to experiment new planning and coordination strategy and so on. At this time, the primary implementation has been done. As a future work, we have to evaluate the system in practical and to introduce another sensor such as omni-directional camera and laser range sensor, etc. On the other hand, the system has been yet considered physical effects in the simulation, nevertheless, we plan to introduce simple method by using collision detection.

The authors would like to express their gratitude to the Information-technology Promotion Agency (IPA), Japan for sponsorship of this project and Professor Takaichi Yoshida for supporting.

References

[1] Kentaro Oda, Takeshi Ohashi, Takeshi Kato, Toshiyuki Ishimura, Yuki Katsumi, The Kyushu United Team in the Four Legged Robot League, in Robocup-2002: Robot Soccer World Cup VI, page.452, 2002

[2] Java 3D(TM) API Home Page. http://java.sun.com/products/java-media/3D/