CogRSim: A Robotic Simulator Software

Yao Shi[1], Chang'an Yi[2,*], Yuguang Yan[1], Deqin Wang[1],
Guixin Guo[1], Junsheng Chen[1], and Huaqing Min[1]

[1] School of Software Engineering, South China University of Technology,
510006 Guangzhou, China
[2] School of Computer Science and Engineering, South China University of Technology,
510006 Guangzhou, China
yi.changan@mail.scut.edu.cn, yichangan.ok@gmail.com

Abstract. Robot simulator is highly necessary and useful because it could provide flexibility and repeatability in robotic research, and the user is able to freely exert his imagination to design robot environment and algorithms. This paper presents CogRSim, a robot simulator which offers four features that differentiate it from most existing ones: (1) 3-dimensional scene could be viewed synchronously in 2-dimensional panel, which makes the user easy to observe the interaction process between the robots and environment; (2) user-defined experimental result might be displayed in appropriate style similar to Matlab; (3) the user can adjust the simulation speed, pause, restart, slower or faster; (4) some objects can be assembled into a complex one in arbitrary shape and unified physical attribute. CogRSim's architecture and subsystems are described in detail, and then a navigation experiment is carried out to verify its usability.

Keywords: Robot simulator, Open Dynamics Engine, Irrlicht.

1 Introduction

The final aim of robotics research is to make the real robots work correctly and effectively following the algorithms. However, some characteristics of robot hardware, such as low abrasion resistance, high price, difficult to produce and modify, etc, prevent the researchers from freely carrying out experiments. Simulations are easier to setup, less expensive, faster and more convenient to use, as a result, they are often prior to investigations with real robots. A good robot simulator has at least three advantages: decrease the abrade on the real hardware; the researcher could not only change the design of the robot and environment at any time, but also test any new algorithms or even unimaginable ideas; being a good communication media among people from different locations and research fields.

A central principle of cognitive robotics is that effective systems could be designed by eliminating complex internal representations and focusing instead on the direct

* Corresponding author.

X. Zhang et al. (Eds.): ICIRA 2014, Part II, LNAI 8918, pp. 155–166, 2014.

relation between environmental stimulus and behavior generation. From this perspective, a good simulation experiment must simultaneously provide three kinds of accurate models: the robot's own geometry, kinematics and sensors; the environment; the physics interaction between the robot and environment. Furthermore, the computational power of computers now has been significantly promoted which makes it possible to run computationally intensive algorithms on personal computers instead of special purpose hardware.

Since 1980s, robot simulator has gained more and more attention and a growing number of commercial and open-source software-simulation tools have been designed both at home and abroad. Existing simulators are either expensive or inflexible in task design, many of them do not contain study material for new users and researchers, and all of these are the motivation to develop CogRSim which is designed to be a free, robust and easy-to-use simulation software for robot research and education.

Besides the traditional functionalities, the main contribution of CogRSim contains four aspects, (i) the two-dimensional (2D) display of the three-dimensional (3D) scene in real time, (ii) flexible display of run-time analysis, (iii) liberal control of simulation speed, such as pause, stop, faster, slower and restart, (iv) assemble some objects into a complex one which has unified physical attribute.

The rest of this paper is organized as follows: section 2 discusses the related work of robot simulators especially the most recent and popular ones, section 3 and 4 present the architecture and subsystems of CogRSim respectively, section 5 gives an example to illustrate the usability of CogRSim, section 6 concludes the whole paper and points out the future trend.

2 Related Work

The robotics community has started to devote increasing interest to simulation tools and their applicability, for example, a workshop at the 2009 IEEE International Conference on Robotics and Automation (ICRA) is about the need for open-source robotic simulation software. Moreover, there are also some papers published talking about simulators [1], [2], [3], [4].

Webots is a commercial development environment used to model, program and simulate mobile robots [5], [6]. With Webots the user can design complex robotic setups with any number of similar or different robots in a shared environment, and the user can choose or set the properties of each object. The robot controllers can be programmed with the built-in IDE (Integrated Development Environment) or with third party development environments. Furthermore, the provided robot libraries enable the user to transfer the control programs to several commercially available real mobile robots. Now, Webots is used by over 1137 universities and research institutes worldwide.

Gazebo is a multi-robot simulator for outdoor environment [7]. With Stage, the user is able to simulate a population of robots, sensors and objects, but does so in 3D world. Because it includes an accurate simulation of rigid-body physics, it could generate both realistic sensor feedback and physically interactions between objects.

Gazebo is developed in cooperation with the Player and Stage projects, and it is compatible with Player. The Player and Stage projects have been in development since 2001, during which time they have experienced wide spread usage in both academic and industry fields. Player provides network-transparent robot control, and Stage is a simulator for large populations of mobile robots in complex 2D domain and is quite capable of simulating the interactions between robots in indoor environments.

Microsoft Robotics Developer Studio (MRDS) is a Windows-based 3D simulator, and it can be used by both professional and non-professional developers as well as hobbyists [8]. MRDS is freely available but not open source, its programming environment is .NET-based and the user could build robotic applications across a variety of hardware equipment. The Visual Simulation Environment, a key part of MRDS, uses Microsoft XNA Framework to render the virtual world and NVIDIA PhysX to approximate interactions between objects within the world. MRDS's current version includes support for both real and simulated Kinect sensor which can be used for navigation and interaction with people.

In fact, some tools that are not dedicated for robot simulation could also provide similar functionalities. For example, Robotics Toolbox encapsulated in Matlab could simulate the kinematics, dynamics and planning for robotic application [9], Robot Operating System (ROS) which is a software framework for robot software development might also offer simulation tools [10]. Furthermore, some simulators are even specific for a particular kind of robot such as Webots for NAO [11] and simulator of iRobot [12], the stages to develop such a robot application include model, program, simulate and finally transfer to a real robot.

3 Architecture of CogRSim

To develop a robot simulator, two basic tools are always elaborately selected. The first is physics engine to simulate rigid body dynamics and features such as joints, collision detection, mass and rotational functions, the other is rendering engine to render the robotic models, objects and environment in 3D scene. In our development tools, these two engines are Open Dynamics Engine (ODE) [13] and Irrlicht [14] respectively which are both free and open source, the operation system is win7, the IDE is Visual Studio 2010, the GUI (Graphical User Interface) is developed by Qt Creator [15] , and Matlab version is 2010b. Fig. 1 shows the relationship between these tools, and the arrows represent control flow.

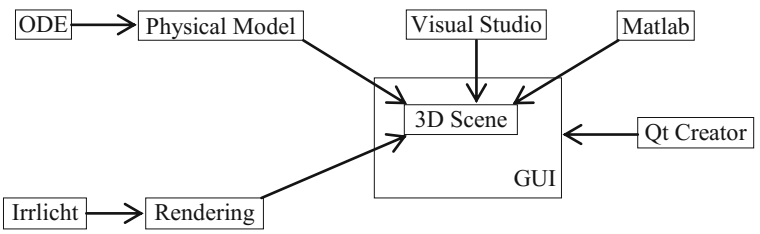


Fig. 1. The relationship between the tools to develop CogRSim

3.1 General Architecture

CogRSim consists of a number of essential modules (subsystems) to make it both easy-to-use and powerful, its general architecture is shown in Fig. 2. Module "World Model" provides the model for robots, objects and other elements in 3D scene, and it is the basis of "Scene Editor" to construct appropriate scene. In 3D scene, an element represents a robot, object, or any other individual thing.

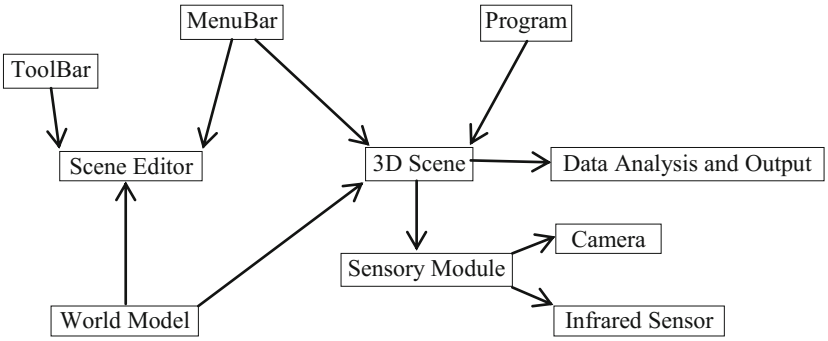


Fig. 2. The general architecture of CogRSim, and the arrows represent data flow

3.2 Module Management

The modules in CogRSim need to communicate with each other, and class "MainManager" is used to manage the communication among them. For each module, there is a pointer inside MainManager to control this module.

Take camera as an example to explain the work flow of MainManager: When method "createCameraAction" sends the signal of creating a camera, MainManager will receive this signal and call method "createCamera" to create a camera, then some parameters will be sent to 3D scene. MainManager has the pointers of 3D scene's display and edit, thus all the work could be finished in MainManager and the consistency could be guaranteed. This work flow can also be described as (1):

$$\text{createCameraAction} \rightarrow \text{MainManager} \rightarrow \text{createCamera} \quad (1)$$

4 Detailed Implementation

4.1 Main Window

CogRSim's main window is shown in Fig. 3, the user can edit the 3D scene and program to test his idea. Now, the "Code Window" depends on Visual Studio to write code for the user. By dragging the mouse or set the corresponding attributes, the users can also change the viewpoint's position, orientation and zoom as shown in Fig. 4, and

this functionality is realized by the main camera in 3D scene. On the other hand, the layout of modules would not be deformed when be dragged, and floating window could be activated, scaled or reopened at any time.

Through ODE, the time flow could be controlled and the simulation speed might be adjusted, for example, if the user needs faster speed instead of higher precision he only needs to decrease the time step. This functionality is illustrated through ControlBar in Fig. 3.

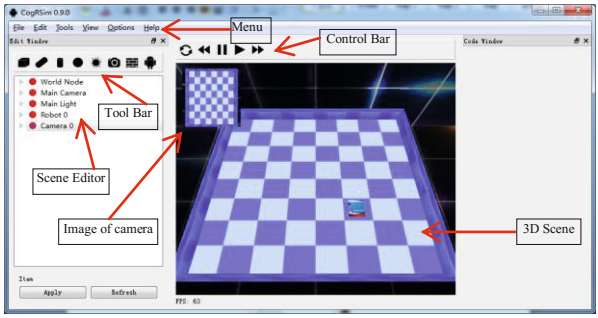


Fig. 3. Main Window and its modules

(a) (b) (c) (d)

Fig. 4. Observe the whole scene in different angles. (a) the buttons are used to change visual angles; (b), (c) and (d) are different angles of the main camera.

4.2 Scene Editor

This part is used to edit for and get current information from the 3D scene. In CogRSim, the user could use the classes TreeView, ToolBar, MenuBar to edit 3D scene, and their relationship is shown in Table 1.

Table 1. The core classes related with Scene Editor

Class Name	Introduction of the class
TreeView	A base class for constructing the tree structure in scene editor
ToolBar	A quick way to add certain kinds of elements for 3D scene
MenuBar	Similar to ToolBar, but could define the attribute values in advance

The result of any operation in ToolBar or MenuBar will display in the tree structure, ToolBar is shown in Fig. 3 and MenuBar is shown in Fig. 5.

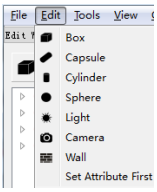
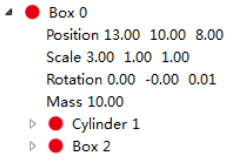


Fig. 5. MenuBar embedded in the menu

To edit 3D scene, the Editor module only sends signals without caring how the signals will be processed, and MainManager will revoke the corresponding modules for further process. This work flow can be described as (2):

$$\text{Editor module} \rightarrow \text{MainManager} \rightarrow \text{3D scene} \tag{2}$$

In TreeView, when the user sends the signal of being selected of a node, the method "refreshNode" will receive this signal and refresh the node information, and the method "itemSelectedSlot" will also receive this signal and then highlight the object or robot in 3D scene. The user can also assemble objects easily, install cameras for objects or robots, as shown in Fig. 6. When deleting the parent node, the children nodes will be deleted together. Furthermore, Light as a shining source could be created in a user-defined position in the 3D scene to send bright.



(a) Assemble three objects in TreeView

(b) The assembled object as a whole

Fig. 6. Assemble some objects into a complex one which has unified physical attribute

4.3 2D Display in Real-Time

When many robots and objects are interacting, the user needs to observe them in a simple form in real time. In this module, all the robots, objects, as well as the trajectory of the robot, are drawn in predefined shape, size, color and name. All the original information are gained from ODE and Irrlicht, and are drawn through Widget of Qt Creator. Class "QPainter" could draw any kind of shape with Qpen or QBrush, and it has various methods such as setBrush(), setPen() and drawRect().

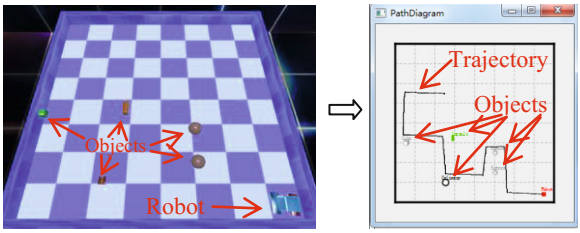
To draw the trajectory of the robot, we use an object called "robot_path" to store its historical coordinates and an integer variable to control the repaint time interval, the

initial position of the robot could also be gained and stored in "robot_path". To make the 2D display correctly, we use some "Managers" to manage the robots and objects, as shown in Table 2.

Table 2. Managers to manage the elements in 3D scene

Class name	Function
RobotManager	Gain and store the coordinates of the robot
ObjectManager	Read and store the categories and positions of objects
WallManager	Store the walls in 3D scene

For data obtain, each kind of object has a link list, for example, object list, robot list, obstacle list, etc. From the link list, the system could get the coordinate directly in real time no matter how large or complex the 3D scene is. With data in the link list, the run time information could be re-displayed again and again. Furthermore, the management of all the elements in 3D scene is through the same source, and the information could be gained in real time and in high precision. Fig. 7 shows the 3D scene and its corresponding 2D display, the dark broken line represents the robot's trajectory and the objects are also drawn in predefined style.



(a) 3D scene (b) 2D display

Fig. 7. 2D display of 3D scene. The objects in (a) are shown in (b), and the robot's trajectory is also drawn.

4.4 Camera

Camera is an important sensory for a robot to gain outside information, just like the eyes of human beings. In order to view the world in multi-angle, there should be many cameras in different locations and angles. In CogRSim, the user could fix a camera in any position of the robot or object, and the camera will move together with it and could gather dynamic information. The image captured by the camera is displayed in real time, but the user can choose whether to be shown, as well as select and drag any image in the 3D scene.

To realize the above functionalities, three classes, Camera, CameraItem and CameraManager, are used, and they are described in Table 3 in brief.

Table 3. Three classes for camera subsystem

Class name	Introduction of the class
Camera	Attributes and basic methods of a camera, for example, id, position, whether being chosen, whether to be shown, information save.
CameraItem	Store the pointer of each camera, and could get the pointer of each camera object through its id.
CameraManager	Manage all the cameras, the construction and deletion of each camera, the render of the camera information by priority when overlapped.

One example of multi-camera is shown in Fig. 8. There are three cameras in different locations, and the top left picture is from the camera under the robot which could capture the floor and wheels.

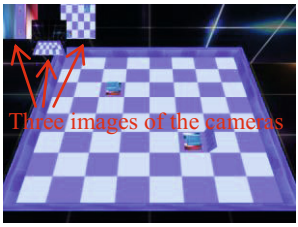


Fig. 8. Three cameras in different position and direction

4.5 Infrared Sensor

Infrared sensor is usually set in front of the robot to detect obstacles, and then the robot could decide whether to avoid. Based on irrode which is the combination of Irrlicht and ODE, the infrared sensor in CogRSim is realized to detect the distance of an obstacle in a given direction. There are three classes, RayListener, RayListenerItem and RayListenerManager, to implement that functionality, and they are described in Table 4 in brief.

Table 4. Three classes to realize Infrared Sensor

Class name	Introduction of the class
RayListener	Inherits from IIrrodeEventLsitener of Irrlicht, could handle the event of object detection.
RayListenerItem	Could store the infrared sensor
RayListenerManager	Could manage the infrared sensor

The infrared sensor and its monitor have a one-to-one relationship, as a result, one monitor for a certain infrared sensor will not monitor others. Fig. 9 shows the infrared sensor of a robot, there are two rays emitted from the robot to detect obstacles.

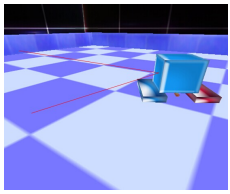


Fig. 9. Robot's infrared sensor and the two rays in red

4.6 Data Analysis and Display

Many kinds of data will be created during experiment, for example, robot speed, object location, images of cameras. The user needs to write code to gather and analyse specific data according to his purpose, and show the result in the right form. In CogRSim, this module provides interfaces to draw different kinds of figures such as curve, box plot and curved surface.

To realize this module is by mixing the programming of C++ and Matlab: first program the script to realize the draw of these figures, then compile the script to DLL (Dynamic Link Library) which could be invoked and further encapsulated by C++. This module is independent and transplantable, and the programmer only needs to invoke these interfaces to draw figures. Fig. 10 shows a typical example of robot speed.

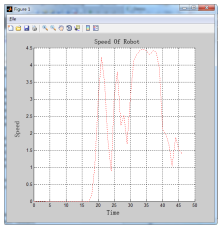


Fig. 10. Curve of robot speed from CogRSim

4.7 World Model

The interaction among robots and objects should meet the principles of kinematics, dynamics and Collision detection, and ODE is chosen to make accurate physical effect. The rigid bodies are generated by composing primitive shapes such as cubes, spheres and cylinders, and each one has been assigned mass, friction, color and texture. In ODE, motor and servo could provide the motivation power, joints could connect bodies together and form complex static ones. There are several types of

joints such as universal joints, ball and socket joints, and hinge joints, the users are able to create robots or complex objects via these components.

To build a simple wheeled robot as shown in Fig. 11, it only needs one box as the main body, two cylinders as wheels where each cylinder is associated with the box by a joint, then motor is added to the joint to provide power. However, to build a table which is a static object without any motor ability, it only needs several boxes to glue with joints.

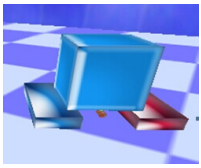


Fig. 11. A two-wheel robot which has power to move and rotate

5 Experimental Validation

In order to illustrate the applicability of CogRSim, a navigation task is carried out on a mobile robot. The task is shown in Fig. 12(a), the thick black lines represent walls and they divide the 8-by-8 maze into two rooms (A, B). Each two neighboring grids are reachable if there is no wall between them. There are four candidate trigger grids (T_1, T_2, T_3, T_4) in room A and four candidate goal grids (G_1, G_2, G_3, G_4) in room B, the start place is a random grid in room A. "trigger" means that when the robot arrives that grid, the two doors will both open immediately as shown in Fig. 12(b). Obstacles will appear dynamically and randomly during the whole task execution, some could be rolled away while the others could not. The robot's task is to first navigate from the start grid to a trigger to make the doors open, then pass a door, and finally to the goal, all following the shortest routine.

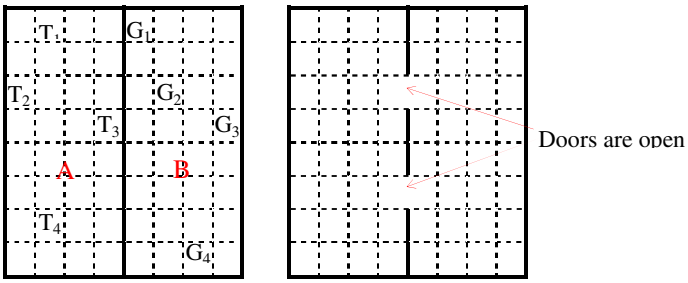


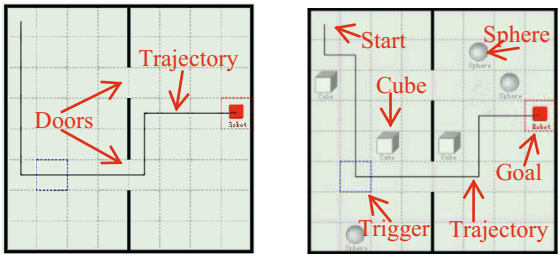
Fig. 12. The robot's task in a maze environment. When the robot navigates to a trigger, the two doors will both open immediately as shown in (b).

According to the task, we set up experimental environment in Fig. 13. The two-wheel robot has four primitive actions, *North*, *South*, *West* and *East*, and they are always executable.

(a) the doors are closed (b) the doors are open

Fig. 13. 3D Scene. In (b), the robot has passed the door

The robot's task has been executed for many times, and one test's result is shown in Fig. 14.



(a) without obstacle (b) with obstacles

Fig. 14. Robot's trajectory is displayed through the module of "2D display synchronously with 3D scene". (a) the robot finishes its task without obstacle, (b) is the final snapshot of the environment, the spheres could be rolled away by the robot while the cubes should be avoided.

6 Conclusions and Future Work

Simulation is well established in robotics and nearly every researcher benefits from it. This paper presents the overview of CogRSim, a software developed for the simulation of robotic models, sensors, and control in virtual environment. The existing functionalities include world model, scene edit, camera and infrared sensor management, user-defined information analysis, and the control of experiment rhythm. CogRSim has been used to carry out some experiments such as navigation and transportation, and they prove to work well.

In the future, at least three problems will be considered. The first is the cloud storage and sharing, through which the users could store, fetch and run 3D scenes in different hardware, and this is the exploration to study cloud robot in reality [ ]. The second is algorithm library, which could be used and updated by users. The last one is code editor, which integrates the functions of edit and debug.

Acknowledgments. This work is supported by National Natural Science Foundation of China (Grant No. 61372140), and National Training Programs of Innovation and Entrepreneurship for Undergraduates (Grant No. 201310561032). Thanks to open source software Irrlicht and Open Dynamics Engine.

References

1. Staranowicz, A., Gian, L.M.: A survey and comparison of commercial and open-source robotic simulator software. In: Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments. ACM (2011)
2. Michal, D.S., Etzkorn, L.: A Comparison of Player/Stage/Gazebo and Microsoft Robotics Developer Studio. In: 49th Annual Southeast Regional Conference, pp. 60–66. ACM (2011)
3. Reckhaus, M., Hochgeschwender, N., Paulus, J., et al.: An overview about simulation and emulation in robotics. In: Proceedings of SIMPAR, pp. 365–374 (2010)
4. Carpin, S., Lewis, M., Wang, J., et al.: USARSim: a robot simulator for research and education. In: 2007 IEEE International Conference on Robotics and Automation, pp. 1400–1405 (2007)
5. Webots: robot simulator, `http://www.cyberbotics.com/`
6. Michel, O.: WebotsTM: Professional mobile robot simulation. arXiv preprint cs/0412052 (2004)
7. Player Project, `http://playerstage.sourceforge.net/`
8. Microsoft Robotics Developer Studio 4, `http://www.microsoft.com/robotics`
9. Corke, P.: A robotics toolbox for MATLAB. IEEE Robotics & Automation Magazine 3(1), 24–32 (1996)
10. Powering the world's robots, `http://www.ros.org`
11. Aldebaran Robotics, `http://www.aldebaran.com/`
12. iRobot Create Programmable Robot, `http://www.irobot.com/create`
13. Open Dynamics Engine, `http://www.ode.org/`
14. Irrlicht Engine, `http://irrlicht.sourceforge.net/`
15. Qt Project, `http://qt-project.org/`