grSim – RoboCup Small Size Robot Soccer Simulator

Valiallah Monajjemi[1], Ali Koochakzadeh[1], and Saeed Shiry Ghidary[2]

[1] Parsian Robotic Center, Electrical Engineering Department, Amirkabir University of Technology
[2] Computer Engineering and Information Technology Department, Amirkabir University of Technology

Abstract. Realtime simulation of RoboCup small size soccer robots is a challenging task due to the high frame rate of input vision data and complex dynamic model of robots. In this paper we describe a new multi-robot 3D simulator for small size robot soccer domain named 'grSim'. In order to decrease the model complexity and increase simulation speed, a simplified dynamic model for omni wheels is implemented. grSim has a distributed architecture, feature-rich user interface and supports all aspects of a small size robot soccer game, thus it can completely replace all hardware used by teams during software development. grSim can help software/AI developers design smarter SSL robot teams.

Keywords: Multi Robot Simulator, RoboCup Small Size League, Omni-directional robot modeling.

1 Introduction

Developing artificial intelligence software for mobile robots is one of the most challenging tasks in the process of designing an intelligent robot. Robot software development usually requires a full functional real robot, however due to the hardware problems which experimental robots always suffer from, it's hard to design software during hardware development process. In addition, when a full functional real robot exists, constraints like cost, maximum operation time and possible damages slow down the software development process. Robot simulators can overcome such problems.

A robot simulator is used for developing software without depending "physically" on the actual robot, thus saving cost and time. In advanced simulators, robots and objects are modeled as rigid bodies in a virtual world. After receiving commands from clients or controllers, a physics engine simulates actions and sends simulated robot perception data back to the client. The virtual world can also (but not necessarily) be visualized using a two-dimensional or three-dimensional graphics layer.

Small Size robot soccer focuses on the problem of intelligent multi-agent cooperation and control in a highly dynamic environment with a hybrid centralized/distributed system [3]. Small Size League (SSL) is one of the main leagues

T. Röfer et al. (Eds.): RoboCup 2011, LNCS 7416, pp. 450–460, 2012.

of international RoboCup [2] competetions. In a small size robot soccer game, two teams of five autonomous four wheeled omni-directional robots play against each other in a 6.05m x 4.05m field with an orange golf ball. A global vision system (which is also a shared system) with two overhead cameras and an off-field PC calculates localization data and sends it to each team's AI computer (which is also off-field) via network connections. After a series of high level decision making and low level control algorithms, commands are sent to the robots using wireless communication.

SSL robots must fit inside a cylinder with radius of 9cm and height of 15cm. This size constraint, in addition to high speed of the robots (2 m/s and above) and the fact that there must be five robots to form a team, makes the design and development of the robots a complex and time-consuming task. Software development for SSL robots would be a hard and ineffective task without help from realistic multi-agent simulation environments.

SSL software developers always have suffered from lack of a realistic simulator. Existing multi-purpose simulators are not applicable for this field, therefor designing a realistic simulation environment would be a great help to small size soccer robots AI developers. The main problem in order to simulate small size soccer robots are their complex dynamic model caused by their four wheel omni-directional movement structure.

UberSim [6][10] is a vision centric 3D simulator designed for using in dynamic environment like RoboCup domain. It supports limited numbers of sensors and actuators. Robots are described as a set of C++ classes. The communication between clients and simulator are TCP/IP based. Although The main application of this simulator has been defined on small size soccer robots domain, the simulator is not under active development for years and is somehow outdated. Besides, it is not compatible with new SSL shared vision system, it lacks a powerful and easy to use graphical user interface and run-time configuration panel.

SimRobot [15] is another multi purpose simulator which supports more sensors and actuators. Robots are described using XML description files. It has a rich graphical user interface and it is possible to add some user/world interaction to the simulation environment.

Another multi-purpose simulator is Gazebo [13] [14] which is part of Player / Stage [9] [4] project which supports many types of sensors and actuators, as well as some popular pre-made robots. Robots and environment description is based on XML files. Users can develop controllers using a rich Application Programming Interface (API).

Webots [16] [8] is a commercial Simulator with a rich set of sensors, actuators and pre-made robot models. Webots describes the environments and robots in VRML format. Users can develop robot controllers in almost all popular programming languages such as C/C++,Java,Python,Matlab and Urbi [5] [12].

All aforementioned simulators use Open Dynamics Engine (ODE) [19] as rigid body dynamics engine and Open Graphics Library (OpenGL) [1] for visualization.

The ability to support a broad range of sensors, actuators and configurations, adds some level of complexity to all aforementioned simulators, as a result, their performance drops significantly in multi robot environments like small size soccer robot domain with 10 agents inside. Dramatically all those simulators, except Ubersim, have a complex model for omni wheels which slows down the simulation speed for small size soccer robots.

A feature-rich user interface with abilities like run-time configurations in addition to easy robot and ball manipulation can reduce time and cost and increase efficiency during development of multi robot systems' low level and high level algorithms; a feature which is missing (or is very hard to implement) in the those simulators.

In order to overcome such problems and develop a state of the art 3D simulation environment for small size soccer robot domain, we developed a brand new simulator named "grSim" which will be described in detail in rest of this paper.

2 Overview

grSim is a multi-robot simulation environment designed specially for RoboCup small size soccer robot domain. It is able to completely simulate and visualize a robot soccer game with full details. Teams can communicate with the simulator in the same way they communicate with real world, except the commands should be sent to the simulator via network instead of radio connections to the robots. In this way they can use this simulator as a powerful tool to test and develop low level control and high level decision making algorithms without the need for real robots.

grSim is developed in C++ programming language with Qt framework [17]. Qt is a cross-platform application development framework which makes it easy to develop cross-platform GUI applications. The simulator has been developed for GNU/Linux like operating systems, but it can be easily ported to other operating systems as well.

In each cycle, two data packets are received from clients (artificial intelligent softwares developed by teams) via network connections. These packet contain desired control commands for each robot's actuators (wheels, kickers and the spinner [details in section 3.1]). After interpreting the information, the appropriate commands are fed to the physical layer of the program.

The physical layer uses Open Dynamics Engine (ODE) [19] physics engine. This library supports many types of rigid bodies, joints and collision detection functions. It can also simulate friction and bounciness. Each robot consists of some basic objects and actuators connected together with joints.

The visualization layer of grSim uses Open Graphics Library (OpenGL)[1] API. Due to the native support of hardware acceleration in OpenGL, this layer renders the virtual environment in an acceptable frame rate and in an aesthetic way.

At the end of each cycle, localization data (the current state of the objects which were calculated in physical layer) are sent back to the clients. The format

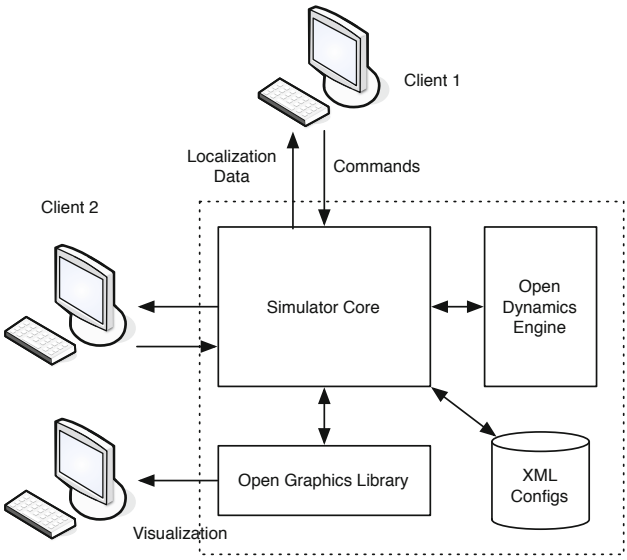

Fig. 1. Overview of the system

used for encoding data is the same format used by small size league's shared vision system, named SSL-Vision. SSL-Vision[21] is the core software of league's shared vision system. As grSim and SSL-Vision use similar data format, clients do not sense any difference between data sent from shared vision system and data sent by grSim, thus no extra effort is needed to convert localization data.

Figure 1 shows an overall view of the simulator's structure.

3 The Simulator

The physical environment consists of 10 robots (5 for each team), a ball, field goals, walls, ground and sky. The dimensions and properties of all objects can be modified in run time. The default values are optimized to comply with latest Robocup SSL rules [7].

3.1 Modeling the Robots in ODE

Each robot consists of four omni wheels, a linear kicker, a chip kicking device and a spin back device. The robot's chassis is a simple cylinder. As the ODE library does not support cylinder with cylinder collision, a dummy sphere is bounded in each cylinder which can only collide with other spheres and cannot collide with any other object. In Figure 2 a small size robot model is depicted.

Wheels. In grSim each robot has four attached wheels with configurable structure. The wheels are solid cylinders attached to the robot's chassis with an ODE's

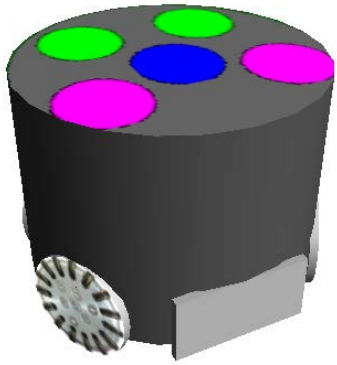

Fig. 2. Robot model in grSim

Hinge joint. In order to rotate the wheels an angular motor with configurable limited torque is attached to each joint.

Omni directional robots like small size soccer robots, use a special type of wheels called omni (or poly) wheels. In omni wheels there are number of sub-wheels around the circumference which are perpendicular to rolling direction. Sub-wheels help the wheel to side-slide.

Physical modeling of sub-wheels increases the model complexity and is a major cause for simulator deficiency. To overcome this problem we implemented a special model for these wheels inspired by [6]. In this model the wheels have a configurable friction with the ground surface. This friction is different for tangential and perpendicular directions (figure 3). Using this technique, the complexity of robot's physical model is decreased which results in less resource usage and higher simulation speed.

Kicker and Spin-Back Device. Each SSL robot is equipped with a special mechanism to kick the ball. This mechanism will let the robot make a direct or a chip kick. In order to simulate the kicker, a solid cube is attached to each robot's front face. Whenever the kick command is received and the ball is in touch, a proper force will be applied to the ball. The magnitude and type of this force are configurable.

Small size soccer robots use a special actuator in front of their robots in order to manipulate the ball while moving. This device, called spin-back or spinner, helps the robot not to lose the ball's possession while moving. In order to simulate this device, the kicker's solid cube applies a torque to the ball when it touches it. In this way the ball rolls around itself and moves backwards, thus sticks to the robot. In real robots, because of the specific shape of the spinner, ball's side movement during spin back does not happen. In order to avoid ball's side movement, grSim uses the same method discussed in section 3.1 between ball and the ground, i.e. defining different friction coefficients for tangential and

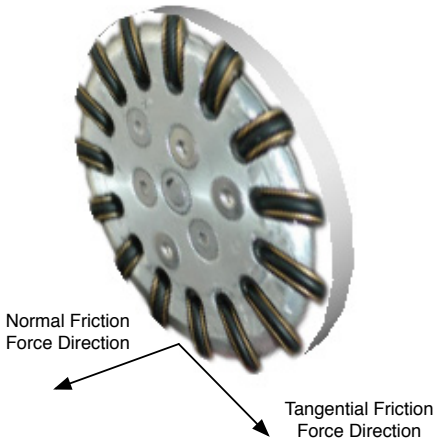

Fig. 3. Omniwheel model in grSim (note that the sub-wheels are textures)

perpendicular directions to the spinner. All of the parameters for the kicker and spinner, like size, maximum torque, maximum force and chip kick angle are configurable.

3.2 Communication

As discussed in section 2 all communications between grSim and clients are network based using User Datagram Protocol (UDP).

Output Packets. The output packet contains the localization data, the position and direction of all ten robots and position of the ball. The packets are generated in the same way that SSL-Vision generates packets. Both SSL-Vision and grSim use Google Protocol Buffer (protobuf) library [11] to encode the data packets. To generate the exact same packet, grSim uses the same protocol configuration file that SSL-Vision uses. The generated packets are sent to the clients with the frequency of 60 times per second, however it can be increased up to 100fps based on simulator rendering frame rate.

To make the simulation output data more realistic, user can specify different kind of noise and disturbances to be added to the localization data. The simulator can apply a two dimensional Gaussian noise to localization data with user specified parameters. It can also add delay to output data to simulate the loop delay which exists in Small Size teams hardware/software architecture. Furthermore, It can add some vanishing effect to output data with user-specified probability. The vanishing effect simulates the temporary loss of an object in vision data. Like other configurations, all these parameters are configurable in run-time.

Input Packets. The simulator receives packets from clients using two UDP sockets, one for the blue team and the other one for the yellow team. Blue and Yellow are standard colors that teams use as the main identifier in SSL matches.

In order to standardize the software's input/output protocols, the commands must be sent using Google Protobuf (section 3.2) encoded data packets. The protocol schema is depicted in Figure 4. The protocol schema describes how the clients must encode their desired control commands before sending data to grSim using Google Protobuf library. The required header and source files to encode data using this schema is included in grSim's software package. According to this schema, in each cycle, the client must first specify the target team, yellow or blue. Next, a series of control commands must be provided for all active robots.

```
message grSim_Robot_Command {
  required uint32 id          = 1;
  required float  kickspeedx  = 2;
  required float  kickspeedz  = 3;
  required float  veltangent  = 4;
  required float  velnormal   = 5;
  required float  velangular  = 6;
  required bool   spinner     = 7;
  required bool   wheelsspeed = 8;
  optional float  wheel1      = 9;
  optional float  wheel2      = 10;
  optional float  wheel3      = 11;
  optional float  wheel4      = 12;
}

message grSim_Commands {
  required double timestamp = 1;
  required bool isteamyellow  = 2;
  repeated grSim_Robot_Command robot_commands = 3;
}
```

Fig. 4. The grSim's Google Protobuf schema for receiving control commands from clients. The protocol documentation and examples can be found in the grSim's documentation and webpage

3.3 User Interface

To ease the software/AI development process, each robot simulator software must be equipped with a good user interface. In order to achieve that, a modern, feature-rich and user friendly Graphical User Interface (GUI) has been developed for grSim. Using the power of Qt framework and its OpenGL integration, the user can modify the objects in the scene, the camera and configurations in an easy way using keyboard or mouse. For example, In order to modify each object's position and orientation, the user can select or move each object by moving the

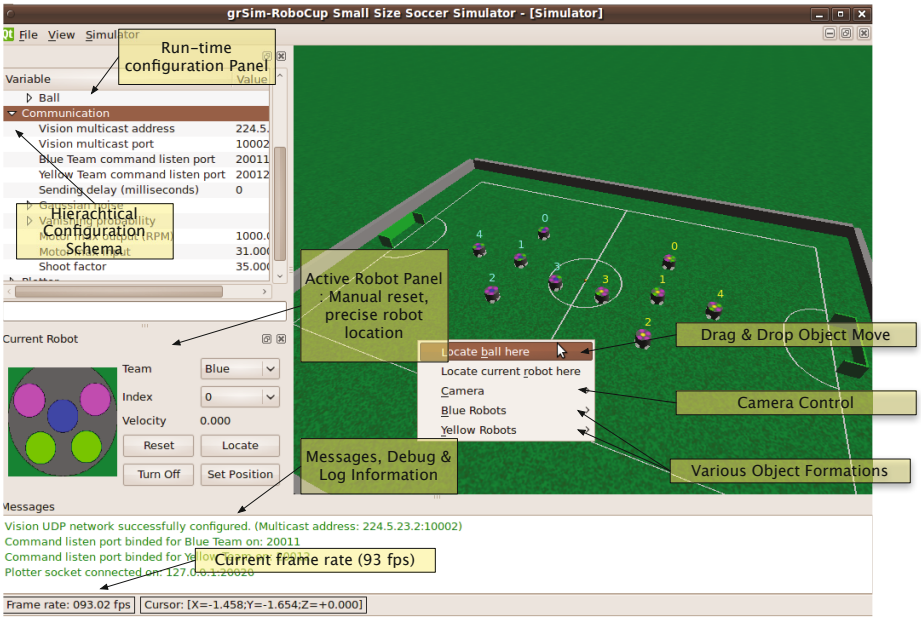

Fig. 5. Features of grSim's Graphical User Interface

mouse and clicking in the scene window. An overview of some UI features are shown in figure 5.

For easy run-time configuration of the simulator parameters, grSim uses VarTypes library[20]. VarTypes is a feature-rich, object-oriented framework for managing variables in C++/Qt4. This library stores all configurable variables and their descriptions in a XML file. It creates a Qt *Widget* for easy modification of variable values during run-time in a thread-safe manner. In this way all of the physical parameters including friction and bounciness of the surfaces, mass of the rigid bodies, actuator parameters and network properties are configurable during program run.

4 grSim in Action

grSim is currently under active development by Amirkabir university's small size soccer robot team named "Parsian"[18]. This simulator plays an important role in the software, control and AI development of aforementioned team.

In order to demonstrate the simulator's accuracy and performance, one robot's simulated motion profile, has been compared to real world data. In the first test, the robot traveled a 4.4m straight line along its local x-axis (its head direction). In the second test, the robot traveled a 3.6m straight line along its local y-axis (perpendicular to its head direction). The position of the robot during the tests are depicted in figures 6, 7 and 8. All the tests were done using 5th generation

Parsian small size robots [18] on a dual core PC with 4GB of RAM running Ubuntu Linux. Both the vision system and simulator's update rate were 62fps during the tests.

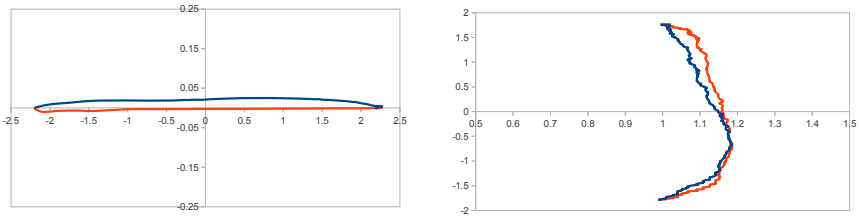

Fig. 6. Comparison between robot's position on the field, while traveling along its head (left) and perpendicular to its head (right). [Blue is from real data]

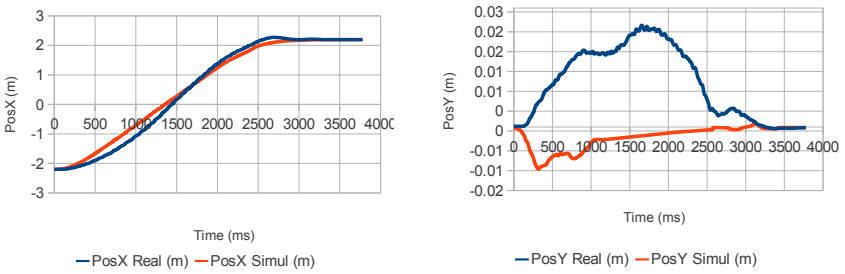

Fig. 7. Comparison between global x and y components of robot's position while traveling along its head direction

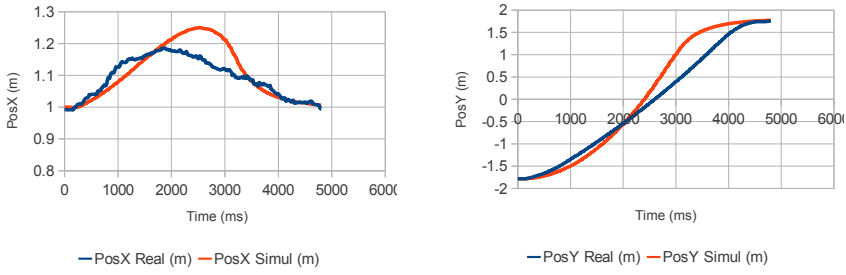

Fig. 8. Comparison between x and y components of robot's position while traveling perpendicular to its head direction

5 Conclusion

In this paper we described grSim simulator, which can simulate and visualize a RoboCup small size soccer robot game in a realistic way and in real-time. By using a simplified robot model it can reach high simulation speeds (60fps or more). The flexible input protocol and SSL-Vision compatible localization data output make it easy to integrate grSim into any existing SSL software chain. grSim's rich user interface makes it an easy to learn, useful tool in AI development process. Our plan is to release grSim as a free and open-source application to RoboCup small size community in near future. Updates, screenshots and videos are available at `http://eew.aut.ac.ir/~parsian/grsim/` .

References

1. OpenGL - The industry standard for high performance graphics (2011), `http://www.opengl.org/` (accessed January 2011)
2. RoboCup - official website (2011), `http://www.robocup.org/` (accessed January 2011)
3. Small Size Robot League - official website (2011), `http://small-size.informatik.uni-bremen.de/` (accessed January 2011)
4. The Player Project - free software tools for robot and sensor applications (2011), `http://playerstage.sourceforge.net/` (accessed January 2011)
5. Baillie, J.: Urbi: Towards a universal robotic low-level programming language. In: 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005), pp. 820–825. IEEE (2005)
6. Browning, B., Tryzelaar, E.: Übersim: A multi-robot simulator for robot soccer. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 948–949 (2003)
7. RoboCup Small Size League Technical Committee: RoboCup Small Size League Rules (2011), `http://small-size.informatik.uni-bremen.de/rules:main` (accessed January 2011)
8. Cyberbotics: Webots, fast prototyping and simulation of mobile robots (2011), `http://www.cyberbotics.com/` (accessed January 2011)
9. Gerkey, B., Vaughan, R., Howard, A.: The Player/Stage project: Tools for multi-robot and distributed sensor systems. In: Proceedings of the 11th International Conference on Advanced Robotics (January 2003)
10. Go, J., Browning, B., Veloso, M.: Accurate and flexible simulation for dynamic, vision-centric robots. In: Proceedings of International Joint Conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2004 (2004)
11. Google Inc.: Protocol buffers - google's data interchange format (2011), `http://code.google.com/p/protobuf/` (accessed January 2011)
12. Gostai Ltd.: Urbi - The universal platform (2011), `http://www.gostai.com/urbi.php` (accessed January 2011)
13. Koenig, N., Howard, A.: Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004), vol. 3, pp. 2149–2154. IEEE (2005)
14. Koenig, N., Howard, A.: Gazebo - 3D multiple robot simulator with dynamics (2011), `http://playerstage.sourceforge.net/gazebo/gazebo.html` (accessed January 2011)

15. Laue, T., Spiess, K., Röfer, T.: SimRobot – A General Physical Robot Simulator and its Application in RoboCup. In: Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) RoboCup 2005. LNCS (LNAI), vol. 4020, pp. 173–183. Springer, Heidelberg (2006)
16. Michel, O.: Cyberbotics Ltd. Webots TM: Professional mobile robot simulation. International Journal of Advanced Robotic Systems 1(1), 39–42 (2004)
17. Nokia Inc.: Qt - A cross-platform application and UI framework (2011), `http://qt.nokia.com/` (accessed January 2011)
18. Poorjandaghi, S., Monajjemi, V., Mehrabi, V., Nabi, M., Koochakzadeh, A., Atashzar, F., Omidi, E., Pahlavani, A., Sheikhi, E., Bahmand, A., Mohaimanian, M., Saeidi, A., Shamipour, S., Karkon, R.: Parsian - Amirkabir university of technology RoboCup small size soccer team. Team Description Paper for RoboCup (February 2011)
19. Smith, R.: ODE - Open Dynamics Engine (2011), `http://www.ode.org/` (accessed January 2011)
20. Zickler, S.: Vartypes - A feature-rich, object-oriented framework for managing variables in C++ / QT4 (2011), `http://code.google.com/p/protobuf/` (accessed January 2011)
21. Zickler, S., Laue, T., Birbach, O., Wongphati, M., Veloso, M.: SSL-Vision: The Shared Vision System for the RoboCup Small Size League. In: Baltes, J., Lagoudakis, M.G., Naruse, T., Ghidary, S.S. (eds.) RoboCup 2009. LNCS (LNAI), vol. 5949, pp. 425–436. Springer, Heidelberg (2010)