Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator

Nathan Koenig, Andrew Howard
Robotics Research Labs, University of Southern California
Los Angeles, CA 90089-0721, USA
Email: nkoenig@robotics.usc.edu, ahoward@robotics.usc.edu

***Abstract*— Simulators have played a critical role in robotics research as tools for quick and efficient testing of new concepts, strategies, and algorithms. To date, most simulators have been restricted to 2D worlds, and few have matured to the point where they are both highly capable and easily adaptable. Gazebo is designed to fill this niche by creating a 3D dynamic multi-robot environment capable of recreating the complex worlds that will be encountered by the next generation of mobile robots. Its open source status, fine grained control, and high fidelity place Gazebo in a unique position to become more than just a stepping stone between the drawing board and real hardware: data visualization, simulation of remote environments, and even reverse engineering of black-box systems are all possible applications.**

Gazebo is developed in cooperation with the Player and Stage projects [1], [2], [3], and is available from `http://playerstage.sourceforge.net/gazebo/gazebo.html`.

I. Introduction

The Player and Stage projects have been in development since 2001, during which time they have experienced wide spread usage in both academia and industry. Player is a networked device server, and Stage is a simulator for large populations of mobile robots in complex 2D domains. A natural complement for these two projects is a high fidelity outdoor dynamics simulator; this has taken form in the Gazebo project.

The necessity of Gazebo has been partially realized with the recent addition of number of all-terrain robotic platforms at the Robotics Research Labs at the University of Southern California. While Stage is quite capable of simulating the interactions between robots in indoor environments, the need for a simulator capable of modeling outdoor environments and providing realistic sensor feedback have become apparent.

Gazebo, therefore, is designed to accurately reproduce the dynamic environments a robot may encounter. All simulated objects have mass, velocity, friction, and numerous other attributes that allow them to behave realistically when pushed, pulled, knocked over, or carried. These actions can be used as integral parts of an experiment, such as construction or foraging.

The robots themselves are dynamic structures composed of rigid bodies connected via joints. Forces, both angular and linear, can be applied to surfaces and joints to generate locomotion and interaction with an environment. The world itself is described by landscapes, extruded buildings, and other user created objects. Almost every aspect of the simulation is controllable, from lighting conditions to friction coefficients.

Following the principles established by Player and Stage, Gazebo is completely open source and freely available (a major advantage over most commercially available packages). As a result, Gazebo has an active base of contributers who are rapidly evolving the package to meet their ever-changing needs.

Gazebo offers a rich environment to quickly develop and test multi-robot systems in new and interesting ways. It is an effective, scalable, and simple tool that has also potential for opening the field of robotics research to a wider community; thus, for example, Gazebo is being considered for use in undergraduate teaching.

This paper describes the basic architecture of the Gazebo package, and illustrates its use and extensibility through a number of user case-studies. We also give some attention to future directions for this package.

II. Player and Stage

Gazebo has been developed from the ground up to be fully compatible with the Player device server. The hardware simulated in Gazebo is designed to accurately reflect the behavior of its physical counterpart. As a result, a client program sees an identical interface to a real and simulated robot. This feature allows Gazebo to be seamlessly inserted into the development process of a robotic system.

Even though it is compatible with Player, Gazebo is not meant as a replacement for the Stage simulator. The complexity of simulating rigid body dynamics coupled with a 3D environment can severely tax even a high performance computer. This has the effect of limiting Gazebo to the domain a few robots, currently on the order of ten. On the other hand, Stage provides a robust and efficient simulator for projects that require large robot populations or do not require the full capabilities of Gazebo.

III. Related Work

Gazebo is far from being the only choice for a 3D dynamics simulator. It is however one of the few that attempts to create realistic worlds for the robots rather than just human users. As more advanced sensors are developed and incorporated into Gazebo the line between simulation and reality will continue to blur, but accuracy in terms of robot sensors and actuators will remain an overriding goal.

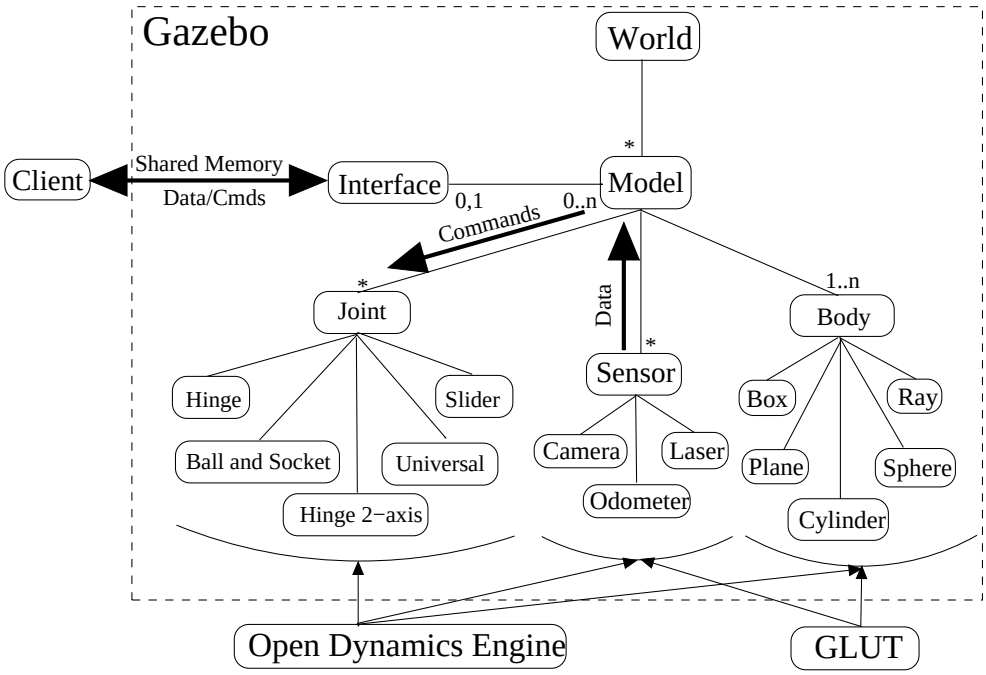

Fig. 1. *General Structure of Gazebo components*

A few notable systems include COSIMIR [4], developed at Festo. This is a commercial package primarily designed for industrial simulation of work flows with robotic systems, but is also applicable to robotic research. COSIMIR has advanced modeling and physical simulation capabilities that go well beyond the capabilities of Gazebo. It incorporates many types of grippers, the ability to program movement in non-robotic models such as assembly lines, and has tools for analysis of the simulated systems. Another commercial package is Webots [5] created by Cyberbotics. Webots allows for the creation of robots using a library of predefined actuators and sensors. When system testing in the simulator is complete, a user can transfer their code to real robots. The principle purpose of Webots is research and development. Cyberbotics is also developing a Player interface for compatibility with a wider range of devices.

Darwin2K [6] and OpenSim [7] represent two open source robot simulators developed along similar lines as Gazebo. Darwin 2K was created by Chris Leger at Carnegie Mellon University as a tool for his work on evolutionary robotics. This simulator accurately models motor and gear heads in fine detail while providing stress estimates on structural bodies. Darwin2K has a strong focus on evolutionary synthesis, design, and optimization and still remains a capable general purpose simulator for dynamic systems. OpenSim, under development by David Jung, is a generic open source robot simulator similar in design an purpose to Gazebo. This simulator makes use of the same third party software packages as Gazebo, and has some attractive features for constructing and debugging articulated joint chains.

IV. Architecture

Gazebo's architecture has progressed through a couple iterations during which we learned how to best create a simple tool for both developers and end users. We realized from the start that a major feature of Gazebo should be the ability to easily create new robots, actuators, sensors, and arbitrary objects. As a result, Gazebo maintains a simple API for addition of these objects, which we term *models*, and the necessary hooks for interaction with client programs. A layer below this API resides the third party libraries that handle both the physics simulation and visualization. The particular libraries used were chosen based on their open source status, active user base, and maturity.

This architecture is graphically depicted in Figure 1. The World represents the set of all models and environmental factors such as gravity and lighting. Each model is composed of at least one body and any number of joints and sensors. The third party libraries interface with Gazebo at the lowest level. This prevents models from becoming dependent on specific tools that may change in the future. Finally, client commands are received and data returned through a shared memory interface. A model can have many interfaces for functions involving, for example, control of joints and transmission of camera images.

A. Physics Engine

The Open Dynamics Engine [8], created by Russel Smith is a widely used physics engine in the open source community. It is designed to simulate the dynamics and kinematics associated with articulated rigid bodies. This engine includes many features such as numerous joints, collision detection, mass and rotational functions, and many geometries including arbitrary triangle meshes (Figure 6). Gazebo

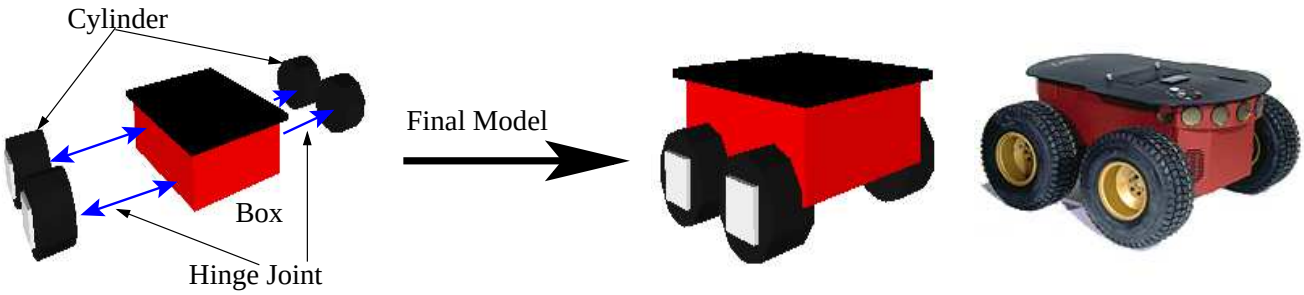

Fig. 2. *Pioneer2 AT construction. Actual Pioneer 2AT shown for comparison.*

utilizes these features by providing a layer of abstraction situated between ODE and Gazebo models. This layer allows easy creation of both normal and abstract objects such as laser rays and ground planes while maintaining all the functionality provided by ODE. With this internal abstraction, it is possible to replace the underlying physics engine, should a better alternative become available.

B. Visualization

A well designed simulator usually provides some form of user interface, and Gazebo requires one that is both sophisticated and fast. The heart of Gazebo lies in its ability to simulate dynamics, and this requires significant work on behalf of the user's computer. A slow and cumbersome user interface would only detract from the simulator's primary purpose. To account for this, OpenGL and GLUT (OpenGL Utility Toolkit) [9] were chosen as the default visualization tools.

OpenGL is a standard library for the creation of 2D and 3D interactive applications. It is platform independent, highly scalable, stable, and continually evolving. More importantly, many features in OpenGL have been implemented in graphic card hardware thereby freeing the CPU for other work such as the computationally expensive dynamics engine.

GLUT is a simple window system independent toolkit for OpenGL applications. Scenes rendered using OpenGL are displayed in windows created by GLUT. This toolkit also provides mechanisms for user interaction with Gazebo via standard input devices such as keyboards and mice. GLUT was choosen as the default windowing toolkit because it is lightweight, easy to use, and platform independent.

C. The World

A complete environment is essentially a collection of models and sensors. The ground and buildings represent stationary models while robots and other objects are dynamic. Sensors remain separate from the dynamic simulation since they only collect data, or emit data if it is an active sensor.

The following is a brief description of each general component involved in the simulator.

1) Models, Bodies, and Joints: A model is any object that maintains a physical representation. This encompasses anything from simple geometry to complex robots. Models are composed of at least one rigid body, zero or more joints and sensors, and interfaces to facilitate the flow of data.

Bodies represent the basic building blocks of a model. Their physical representation take the form of geometric shapes chosen from boxes, spheres, cylinders, planes, and lines. Each body has an assigned mass, friction, bounce factor, and rendering properties such as color, texture, transparency, etc.

Joints provide the mechanism to connect bodies together to form kinematic and dynamic relationships. A variety of joints are available including hinge joints for rotation along one or two axis, slider joints for translation along a single axis, ball and socket joints, and universal joints for rotation about two perpendicular joints. Besides connecting two bodies together, these joints can act like motors. When a force is applied to a joint, the friction between the connected body and other bodies cause motion. However, special care needs to be taken when connecting many joints in a single model as both the model and simulation can easily loose stability if incorrect parameters are chosen.

Interfaces provide the means by which client programs can access and control models. Commands sent over an interface can instruct a model to move joints, change the configuration of associated sensors, or request sensor data. The interfaces do not place restrictions on a model, thereby allowing the model to interpret the commands in anyway it sees fit.

2) Sensors: A robot can't perform useful tasks without sensors. A *sensor* in Gazebo is an abstract device lacking a physical representation. It only gains embodiment when incorporated into a model. This feature allows for the reuse of sensors in numerous models thereby reducing code and confusion.

There currently are three sensor implementations including an odometer, ray proximity, and a camera. Odometry is easily accessible through integration of the distance traveled. The ray proximity sensor returns the contact point of the closest object along the ray's path. This generic sensor has been incorporated into a Sick LMS200 model to simulate a scanning laser range finder, and also into a sonar array for the Pioneer 2. Finally, the camera renders

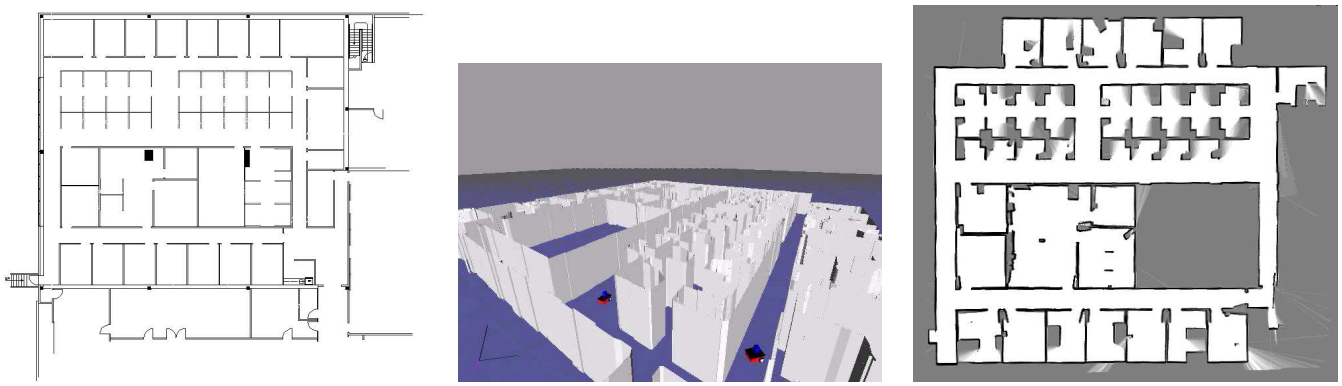

Fig. 3. *Above is a hand-drawn 2d map, the 3D extrusion created from the 2D map, and finally the laser generated map of the real environment.*

a scence using OpenGL from the perspective of the model it is attached to. Currently the camera sensor is used for both a Sony VID30 camera and the ”god’s eye” view of the world.

3) External Interfaces: Gazebo is generally used in conjunction with Player. The Player device server treats Gazebo as a normal device capable of sending and receiving data. From the users point of view, the models simulated in Gazebo are the same as their real counterparts. A second key advantage to this approach is that one can use abstract drivers inside a simulation. For example, it is possible to use Player’s VFH (Vector Field Histogram) [10] or AMCL (Adaptive Monte-Carlo Localization) [11] interchangeably between real and simulated environments.

The interface to Gazebo is not limited to Player alone. The low-level library provides a mechanism for any third-party robot device server (Player or otherwise) to interface with Gazebo. Going even further, a connection to the the library is not even necessary since Gazebo can be run independently for rapid model and sensor development.

Currently the Gazebo library offers hooks to set wheel velocities, read data from a laser range finder, retrieve images from a camera, and insert simple objects into the environment at runtime. This data is communicated through shared memory for speed and efficiency.

D. Construction of Models

Models are currently created by hand. The process starts with choosing the appropriate bodies and joints necessary to build an accurate model in both appearance and functionality. Following Figure 2, we will use the construction of the Pioneer2 AT model as an example. The entire set of bodies for this model encompass four cylinders for the wheels and a rectangular box body.

The next step attaches the bodies together using joints. The end result is a complete physical representation of our model. For the Pioneer2 AT, each wheel has a single axis of rotation. Hinge joints match this requirement perfectly. They are connected to the sides of the rectangular base and the wheels such that the axis of rotation allows for proper wheel spin.

Our Pioneer2 AT model now only lacks an interface for user control. A few functions provided by the Gazebo library resolve this issue, and allow a user to apply velocity changes to the wheels and retrieve odometric data. The Pioneer2 AT model base can also be retro-fitted with any number of devices such as a sonar ring, gripper, and laser range finder.

V. Case Studies

Although still a young application, Gazebo has been used in many interesting ways that have been unavailable in other simulators. These techniques can be generally characterized by an improved robot development process. The first case increased the efficiency at which a complete robot system is built, another used Gazebo for greater safety while demonstrating a useful technique at reverse engineering a robot system, and the final two demonstrate the use of Gazebo in developing and refining new algorithms.

It should be noted that Gazebo is only a tool whose extensiblity has allowed other researchers the opportunity to creatively leverage its functionality for improved develpement and data collection.

A. Morphology and Control

The development of new robotic architectures involve a chicken and egg problem. Is it better to develop a control system to match the hardware, or the hardware to match the required controller? A solution to this problem is rapid prototyping where new iterations of both hardware and software are developed and tested in unison. This has the benefit of revealing the strengths and weakness in both parts with minimal loss of time and effort when mistakes occur. Prototyping of this nature has long been used in the software domain, but still remains expensive and slow in hardware. Gazebo solves this problem by providing means to easily change the robot’s morphology while also modifying its software systems.

This design principle was realized in a recent project [12] at USC. The task involved the creation of a six legged walking robot that learned various gaits. The process of designing and building the hardware alone would have consumed numerous months. Instead Gazebo reduced the time frame down to less than two months, during which time both the physical structure and software were modified

and tested in parallel. Clearly, use of this feature allows developers to easily move from basic concepts to real working systems in a short period of time.

Fig. 4. *Above is a real SegwayRMP and its model in Gazebo below.*

B. Reverse Engineering

A recent addition to the Robotics Research Labs at USC has been a Segway RMP. This is a physically powerful device initially intended to transport humans that has been modified to be a robot platform. Over rough terrain, this robot will sometimes topple, and is therefore quite dangerous to both people and the expensive hardware mounted on it. A method to avoid undue damage during software development involves rigorous testing in simulation. However, creating a realistic model of the Segway RMP offers unique challenges. In order to move, the Segway RMP must tilt forward or back to gain acceleration (it is essentially an inverse pendulum). This tilting affects the interactions it has with the world and particularly the data received from any on-board sensors. It is therefore imperative that any model used to simulate the Segway RMP must accurately reflect these characteristics.

To overcome this problem, Marin Kobilarov reverse engineered the controller used on-board the SegwayRMP. This was accomplished by extensively testing the angles of inclination at various accelerations. The results from these tests were used to tune a controller in Gazebo that now successfully models the characteristics of the real Segway RMP (see Figure 4).

Reverse engineering is a common practice when trying to understand how black-box systems operate. In this case it was necessary as a safety precaution. In other situations it may simply be impossible to have a control scheme specified. In the case of human-robot or even robot-animal interaction, models can only be created through observations of their behaviors. Gazebo in turn provides an effective mechanism to implement and test these derived models.

C. New Environments

Many environments in which robots operate are either well studied or carefully constructed. Deploying robots in a never before encountered world may cause unforeseen, and possibly negative, side effects. Lighting conditions, reflective surfaces, and odd objects can all play an effect on how a robot operates. A strategy of online testing can be extremely slow and tedious. Time can be spent much more productively by testing and modifying the robot controllers offline in preparation for the real experiments.

The fine grained control of Gazebo, the ability to extrude 2D images into 3D structures, and terrain generation (Figure 6) allow for the unique ability to hand create rough outlines of a new environment. This technique was recently realized and employed by Andrew Howard, and demonstrated in Figure 3. When moving into a new environment, he first used a sketch of the test site to extruded a model in Gazebo. This simulated environment was then used as one of test environments during the development of his exploration and mapping algorithms. These algorithms were later successfully deployed in the real environment, after which the real maps were ported back into Gazebo for further off-line experiments.

As a result, the development time of the algorithms employed was greatly reduced. Gazebo made it possible to continue experimentation in the environment even after the physical robots were deployed. In this case a simple sketch map created a sufficient testing environment, however greater accuracy can be achieved with access to terrain data. Elevation information collected from satellites, or other means, can be imported along with relevant structures to further blur the line between simulation and the real world. All of this culminates in the ability of Gazebo to reduce development and test time, and even allow experiments to virtually take place in almost any part of the world.

D. Algorithm Design

The design and implementation of new algorithms can be a difficult task that become particularly acute with the lack of convenient test environments. In situations such as this, Gazebo's sensory realism can play a time saving role. Traditionally, development of new algorithms either required custom simulators or direct testing on the hardware; Gazebo's realistic environments and simple interface can drastically reduce the turn around time from a conceptual stage to functional system.

This technique was used in the creation of a new sensing device; a visual bar code reader. The purpose of this device was to identify uniquely marked objects in arbitrary environments (a black and white bar code system was chosen based on its simplicity, Figure 5). Gazebo was used as the sole test platform, and a visual barcode detector was rapidly developed and tested. Following satisfactory performance in Gazebo, the system ran *unmodified* on a real robot with near identical results. This algorithm has since been proven robust in a wide range of real environments.

VI. Limitations

Gazebo has a number of important limitations. While it is designed as an outdoor simulator, the fidelity of this simulation is limited; for example, physics models of soil, sand, grass, and other pliable surfaces normally found in

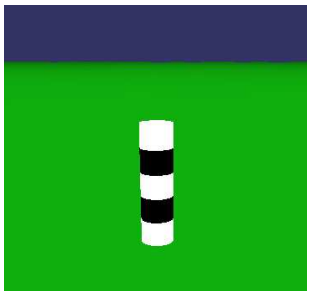

Fig. 5. *A sample bar-code fiducial for visual object recognition.*

nature are well beyond the scope of this project. Added to this list of have-nots are deformable objects, and fluid and thermal dynamics. These features are currently lacking in Gazebo due to their complexity, although some may be added as the need arises.

Distributed computation tops the list of more useful features missing from Gazebo. The amount of computation involved while running such an intensive simulation would greatly benefit from distributed computation. This has the possibility of becoming a reality in the future, but currently presents practical problems due to the used of a monolithic dynamics engine.

VII. Conclusion and Future Work

This paper has explained the design principles of Gazebo and its applicability to the development process of real world robotics. Our software has deliberately maintained a simple yet powerful interface to the underlying physics engine and rendering capabilities while retaining compatibility with the Player and Stage initiatives. This has resulted in a quick and easy adoption of Gazebo by many people, and has the potential to be used in ways never before seen in a simulator.

There however remains much work to be done. While the breadth of robot models has grown rapidly, sensor implementations remain relatively few, primarily due to implementation complexity. Features such as programmable objects (doors, elevators, lights) and multi-level image extrusion will add even more realism and increase the domains in which Gazebo is applicable. Finally, real-time interaction will eventually provide mechanisms for on-the-fly alterations to test scenarios through addition of user specified forces and commands.

In the relatively short life span of Gazebo, we have seen adaption and contributions from other universities and creative uses of Gazebo as not just a simulator but also a safety device. It is our hope that this simulator will continue to provide a valuable tool for current researchers, and also inspire people who lack expensive hardware to perform new and interesting experiments.

Acknowledgments

The authors would like to thank Dylan Shell, Gabriel Sibley, and Marin Kobilarov for their creative use of Gazebo. Many thanks also to the excellent contributions from Pranav Srivastava at the University of Pennsylvania.

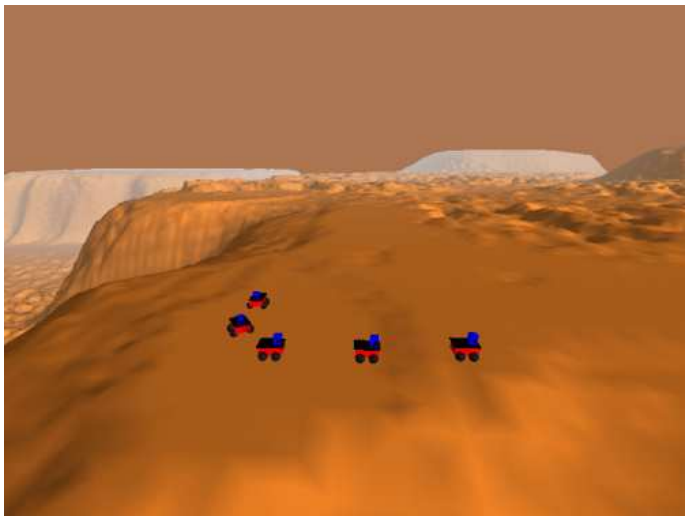

Fig. 6. *Terrain and Pioneer2 AT robots.*

References

[1] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the International Conference on Advanced Robotics*, Coimbra, Portugal, Jul 2003, pp. 317–323. [Online]. Available: http://cres.usc.edu/cgi-bin/print˙pub˙details.pl?pubid=288

[2] B. P. Gerkey, R. T. Vaughan, K. Støy, A. Howard, G. Sukhatme, and M. J. Matarić, "Most valuable player: A robot device server for distributed control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001, pp. 1226 – 1231, (Also appears in Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS at Autonomous Agents 2001, Montreal, Canada, May 29, 2001). [Online]. Available: http://cres.usc.edu/cgi-bin/print˙pub˙details.pl?pubid=91

[3] R. T. Vaughan, B. P. Gerkey, and A. Howard, "On device abstractions for portable, reusable robot code," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, U.S.A, Oct 2003, pp. 2121–2427, (Also Technical Report CRES-03-009). [Online]. Available: http://cres.usc.edu/cgi-bin/print˙pub˙details.pl?pubid=346

[4] http://www.festo.com.

[5] http://www.cyberbotics.com.

[6] C. Leger, *Darwin2K: An Evolutionary Approach to AUtomated Design for Robotics*. Kluwer Academic Publishers, Aug 2000.

[7] http://opensimulator.sourceforge.net/.

[8] http://opende.sourceforge.net/.

[9] http://www.opengl.org/resources/libraries/glut/glut˙downloads.html.

[10] I. Ulrich and J. Borenstein, "Vfh*: Local obstable avoidance with look-ahead verifi cation," in *IEEE International Conference on Robotics and Automation*, San Francisco, CA, Apr 2000, pp. 2505–2511.

[11] D. Fox, "KLD-sampling: Adaptive particle fi lters," in *Advances in Neural Information Processing Systems 14*. MIT Press, 2001.

[12] http://www-robotics.usc.edu/˜dshell/hex/hex.html.