The Doane Roverbot Simulator

Allen Gilbert, Mark Meysenburg
IST Department
Doane College
Crete, NE 68333
[allen.gilbert, mark.meysenburg]@doane.edu

Abstract

As a continuation of Doane College's 2004 Summer Research Program, we undertook the task improving a 3D simulator created to run programs written for a Lego® Mindstorms™ "Roverbot." Our primary goal was to implement some sort of physics engine. We accomplished this by using the Open Dynamics Engine, an open-source solution that provided us with physics modeling and collision detection. We also made improvements to the simulator's Graphical User Interface and camera angle controls. Our new GUI makes it easier for the user to view and interact with the simulator.

Although we made many improvements to it in the summer of 2005, the simulator is far from complete. Still, our efforts have provided a good framework for future improvements, and the experience of developing such a simulator has increased both our knowledge of and respect for software design.

1 Introduction

Over the past few decades, the importance of computer-aided simulation has been firmly established. From advances in space travel to improvements in city traffic control, the ability to simulate real-world events with virtual visualization has proven to be a time and cost efficient way to solve problems. The possibility to reap these analytical benefits of simulation provided excellent motivation for the creation of a Lego® Mindstorms™ (1) Roverbot simulator during our 2004 Summer Research Project. Our vision for the simulator was for it to allow development of Roverbot programs without need for the actual Roverbot hardware.

To better understand our project, one must first be familiarized with the concept behind Lego Mindstorms Robot Kits. In the words of Lego's website,

> LEGO® MINDSTORMS™ lets you design and program real robots that do what you want them to. With the Robotics Invention System 2.0™, the core set of the LEGO MINDSTORMS product range, you can create everything from a light-sensitive intruder alarm to a robotic rover that can follow a trail, move around obstacles, and even duck into dark corners. (1)

The operation of a robot built with a Mindstorms kit is controlled by a microcomputer housed inside an "RCX brick." This yellow brick includes three motor control outputs, three sensor inputs, operational buttons, a digital display, and an infrared receiver. A user can create control programs using bundled Lego software that provides a visual representation of structured programming. Thus, the Lego Mindstorms kits are useful for teaching the basics of programming to beginners.

The Roverbot (pictured in Figure 1) is a simple robot with two different kinds of sensor inputs. The front of the robot houses two bumper sensors that can detect and prompt a reaction to a collision. Although not pictured below, our Roverbot also houses a front-mounted light sensor that can react to differences of dark and light in its path. Two motors drive both the front and back wheels on each side, providing mobility for the Roverbot. Thus, turning is accomplished by supplying more power to one pair of wheels over the other, or by driving the motors in opposite directions.

Our Roverbot's test environment is a 4 x 8 foot wooden "arena" with one-inch ridges on each side. Figure 2 pictures this environment with obstacles (for use with the bumper sensors) and blue tape (for use with the light sensor).

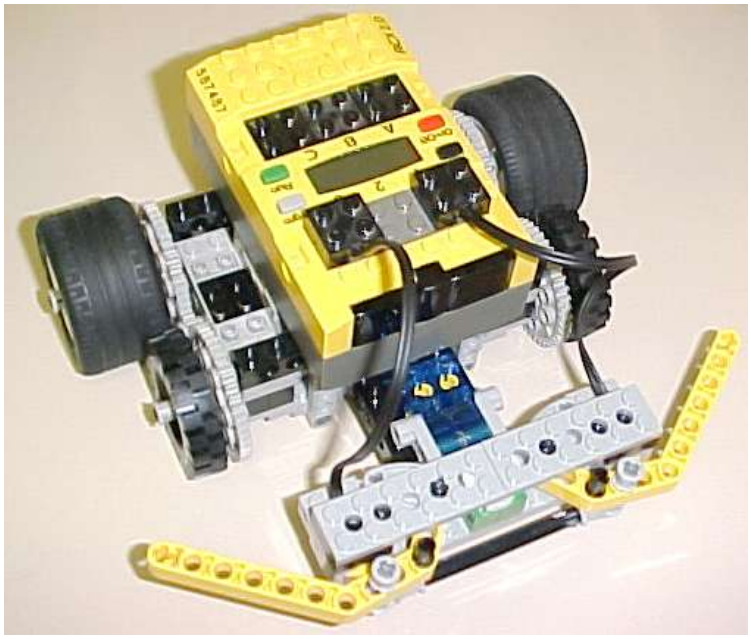

Figure 1: A Real Roverbot

Figure 2: The Roverbot Test Environment

In this paper, we will discuss the basics of the Open Dynamics Engine, our improvements to the Graphical User Interface, and our future plans for the simulator.

2 The Open Dynamics Engine

Initially, the Doane Roverbot Simulator was simply a framework for a simulator. It could display models of the Roverbot, parse and execute simple Roverbot programs, and allow the user to manipulate the camera. However, movement of the Roverbot was based on a crude behavioral modeling scheme. In other words, the virtual Roverbot's movements were timing-based, with little regard for physical accuracy. Furthermore, this behavioral modeling scheme did not implement collision detection or variable power levels for the motors. Clearly, the simulator would not be useful unless we could find a way to make the Roverbot move and behave in a more realistic way.

The idea of creating a physics engine was one alternative. However, authoring such an engine would require an understanding of physics that we did not posses. So, rather than try to reinvent the wheel, we decided to search for an open source physics engine. After a little investigation, we came upon ODE: the Open Dynamics Engine. In the words of its author, Russel Smith,

> "ODE is an open source, high performance library for simulating rigid body dynamics. It is fully featured, stable, mature and platform independent with an easy to use C/C++ API. It has advanced joint types and integrated collision detection with friction. ODE is useful for simulating vehicles, objects in virtual reality environments and virtual creatures. It is currently used in many computer games, 3D authoring tools and simulation tools" (2).

3 ODEJava

Because our simulator was written in Java and Java3D (3), we were able to utilize ODEJava, a Java binding for ODE's C/C++ libraries. The binding not only allows access to low-level ODE "primitives," but it also presents a higher level, truly object-oriented interface to the capabilities of ODE. The required native binaries are available in pre-compiled versions for Linux, Windows, and OS X (2).

In ODE / ODEJava, objects are represented as primitive shapes connected by joints. Each shape has size, mass, and position attributes. There are several types of joints that can be used to connect shapes, including ball and socket, hinge, slider, and fixed or immobile joints.

Figure 3 shows the shapes that make up our ODE Roverbot. In this first use of ODE in our simulator, we kept the ODE Roverbot model as simple as possible. The single large yellow box represents the RCX brick and the driving base of the Roverbot. The smaller white boxes represent the center support structure of the Roverbot's bumper mechanism. The angled yellow shapes represent the bumpers themselves. The gray wheels of the Roverbot are modeled as spheres, rather than cylinders.

All of these shapes are connected together using ODE fixed and hinge joints. The

bumper components are attached to the chassis with fixed joints, which makes them immobile. The wheels are attached to the chassis with hinge joints. These joints are configured so that the wheels only rotate around their axes.

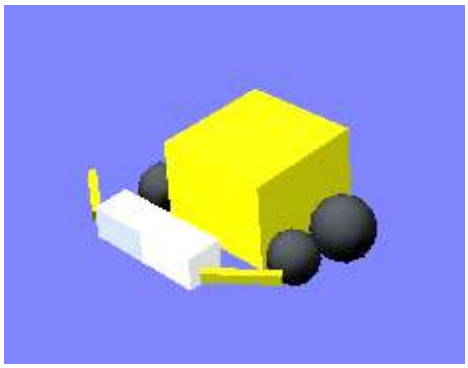

Figure 3: ODE Roverbot model

Once we created our ODE model of the Roverbot, it was fairly straightforward to make our simulated Roverbot move in a realistic manner. Basic movement in the simulator is now achieved by setting the angular velocities of the ODE spheres that represent the Roverbot's wheels. We measured the angular velocities of a real Roverbot's wheels for each of the possible motor power levels. These measurements are stored in a lookup table and applied, along with direction of rotation, to the ODE spheres. The commands that change the power levels and directions of the wheels come from the Roverbot program being interpreted by the simulator.

Another benefit of using ODEJava in our simulator is that collision detection is automatically included. The Roverbot, the Roverbot's "world," and the obstacles that are placed in the world are all modeled as ODE objects. So, when objects collide with each other, the ODE system detects the collision and updates object positions as necessary. Thus, we only had to pay attention to collisions with the Roverbot's bumpers, in order to allow the Roverbot program to respond to the touch sensor event. This was easy to do because every collision in the ODE system generates identifying integers. Through a simple "if" structure, we were able to determine when a collision involved one of the bumpers; then a call back into the RCX code interpreter was made. Thanks to ODE, we were able to have collision detection in our simulator basically for free, without having to write any Java3D collision detection code.

A difficulty we encountered while incorporating ODE into our simulator was the lack of clear documentation regarding the ODE / ODEJava software. There are several "magic" parameters that influence the behavior of the simulator in subtle ways, and we are still unsure exactly which values should be chosen for these parameters. We have a strategy for tuning these parameters, however. This strategy is discussed below in the "Future Work" section.

Another difficulty we had with ODE was that the fidelity of the simulation seemed to vary depending on the size of the objects in the simulation. When we were learning how

to use ODE, we were experimenting with relatively large objects. For example, we worked with a simulated car whose dimensions were better expressed in meters than in centimeters. However, once we started implementing the Roverbot model with its smaller dimensions, we found that the ODE simulation would often "depart from reality" in interesting ways. In particular, the Roverbot model would shake, with all of its component parts oscillating randomly, even when it was at rest. Through experimentation, we found that the simulation behaved in a more realistic manner when all of the dimensions were scaled up by a factor of ten.

4 Graphical User Interface

In order to improve the usability of our simulator, we made some changes and enhancements to our Graphical User Interface. Most notably, we added controls for manipulating the camera in a limited way. Prior to this update, our manual camera scheme was confusing and unintuitive because Java3D's default mouse-controlled rotations and translations were uneasy to control. Instead of trying to rewrite those mouse controls, we created a viewing scheme that follows a hemisphere above the Roverbot's world. The user can zoom, rotate, and angle the camera without having to worry about ending up with a view that is upside down or crooked.

Although not all of the associated features are fully functional yet, we added new menu items to our GUI. Users will be able to save simulation states, place rectangular obstacles of a chosen color, and consult a help file. Furthermore, we added a few icons to increase the visual appeal of the simulator.

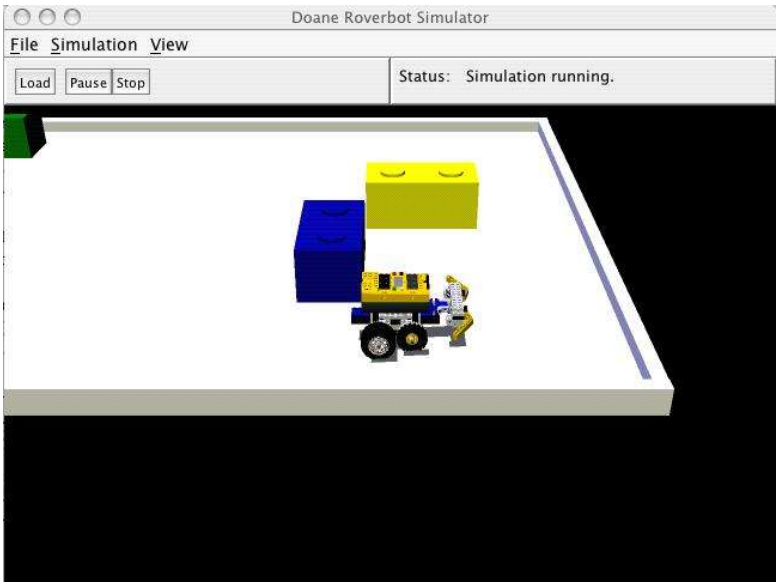

Figure 4: Old Doane Roverbot Simulator GUI

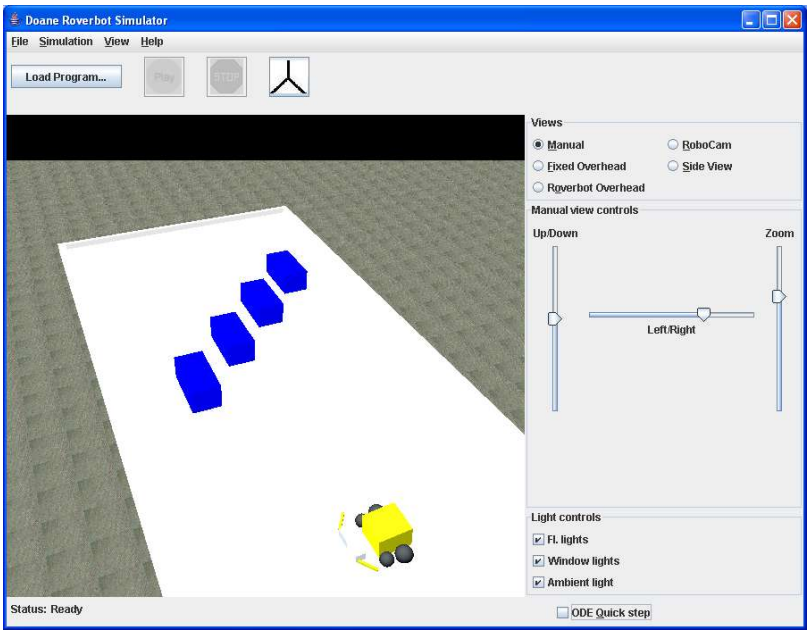

Figure 5: New Doane Roverbot Simulator GUI

5 Future Work

There are several aspects of our simulator that we plan to improve and enhance. Most importantly, we plan on improving the fidelity of the ODE physics model by using a genetic algorithm to fine tune the environmental variables. This should allow us to optimize the "magic" parameters for the best possible representation of reality. The light sensor needs to be incorporated into the simulator, along with the ability to place simulated "blue tape" into the world. We also wish to develop a less restrictive yet intuitive camera scheme, complete with computer-controlled camera angles that smoothly follow the Roverbot. Other planned work involves the ability to load other "worlds" for the Roverbot to operate in. Finally, we wish to create an installer for the simulator so that we can distribute it to others more effectively.

Work on the simulator will continue during the summer of 2006 with another Doane Undergraduate Research project.

6 Conclusion

This project provided us with many opportunities to learn through experience. We were able to learn much about programming in Java and Java3D using the NetBeans development environment. We were all introduced to a very important concept of group programming called "common code ownership," and used the NetBeans CVS (Concurrent Versions System) to keep our files up to date on a server. When using CVS, we learned how to resolve conflicts in different versions of code, how to submit updates,

and how to manage our local development environment.

When implementing the ODE physics model, we were forced to learn independently by reading documentation and exploring examples of ODEjava code. We were reminded of the value of Object Oriented Programming, as it provided an easy way to plug in the Open Dynamics Engine into our existing code. Finally, we gained an appreciation for the concept and practicality of open-source code.

Over all, the experience of creating our simulator was very valuable. Participating in the software design process has increased both our knowledge and respect for software design, 3D modeling, compiler construction, and computer simulation.

References

(1) Lego Mindstorms Website: http://mindstorms.lego.com/eng/products/ris/index.asp. Link active as of 3/23/2006.

(2) ODEjava website: http://odejava.org/OdejavaIntro. Link active as of 3/23/2006

(3) Sun (Java) Website: http://java.sun.com/. Link active as of 3/23/2006.