Realistic Robot Simulator

Nicolas Ward '05
Advisor: Prof. Maxwell

2004.12.01

Abstract

I propose to develop a comprehensive and physically realistic virtual world simulator for use with the Swarthmore Robotics Team's existing robot control software. The simulation software would replace the low-level motor control and sensor library on a robot, connecting its normal sensory inputs to a virtual reality controlled by the user. The goal of this project is to create a testing environment for these robot software modules that does not require the use of any robot hardware, which is becoming unreliable with age. As an additional benefit, we would be able to test the robot's software capabilities in larger, more complex environments than are available here at Swarthmore. Since this project consists entirely of software development, which will be performed using existing department and personal computer equipment, and which will rely solely on open-source software libraries, the projected cost of the entire project is zero.

Introduction

My proposed project, an accurate simulator for an RWI Magellan Pro research robot, is primarily a software design, development, and integration task. For this reason, any potential technical issues will be almost exclusively confined to the code. The technical requirements, software library selections, and expected technical problems are described in further detail in the Technical Discussion.

The nature of this project is such that it is largely linear, although the work can be separated into several phases. Details on what software libraries I need to learn how to use, and what steps I need to take to integrate the simulator with the other modules are given in the Project Plan.

Further information on relevant skills that I will be applying to this project, and my experience with similar work in the past, is given in the Project Qualifications section. This project will be a continuation of my work done for the Swarthmore Robotics Team.

Since this project has no monetary cost, there is no Project Cost discussion in this proposal. The only possible unforeseen costs would be sources of information, namely books or journal articles, the vast majority of which are available for free at the library or online. Scheduling details for my own time, and my best estimate need for advising from Prof. Maxwell, are covered under the Project Plan.

The direct environmental impact of this project is negligible, since the computational resources being used would be powered whether or not I was using them. However, once the simulator is functional, we will not have to power both the robot and the client computer to do some work.

As for ethical considerations, there aren't many to be applied to simulating a robot. In general, however, there are some specific concerns when it comes to mobile robotics. If we get to the point where a robot can traverse a disaster area and find victims, how do we prevent that technology from being adapted to a military use? Searching for enemy combatants in an urban environment would not be a significantly different task. In working in the wide field of robotics, it is important to ensure that the work is not being adapted to evil.

Technical Discussion

Robot Modules

The software currently in use on the Magellans has been in development for several years. Prof. Maxwell has overseen several groups of students who have written most of the code. It is a highly complex system, since it is responsible for controlling the robot, detecting and responding to obstacles in the environment, processing video using computer vision techniques, mapping its trajectory through the environment, and making strategic task-completion decisions, such as goal-based navigation. Each of these subtasks is assigned to a different module, modularity being a desirable feature from a software engineering perspective, since it allows for significantly easier expandability.

The primary set of modules are the Swarthmore Vision Module (SVM), the Swarthmore Mapping Module (SMMD), and the Swarthmore Navigation Module (SNMD or Nav). An autonomous control module, code-named "Pinky", is currently under development by Fritz Heckel '05. All of these modules communicate with each other using CMU Inter-Process Communication (IPC), a message-passing framework developed at Carnegie-Mellon University.

A recent addition to the family is Robomon, a module management daemon developed this past summer by Fritz, with more features being added as part of his Computer Science thesis work.

There are two additional libraries on which these modules depend. The first is Mage, which is used by Nav to communicate with the RFLEX board in the robots over a serial connection. This allows Nav to receive data from the sonar and IR rangefinders, bump sensors, and wheel encoders, and to send motion control commands to the motors. The second library is GCM, the general communications module, which defines a standard format for some of the IPC messages passed between the various modules and those messages used for communication with Robomon.

The final piece of this entire puzzle is the human user interface, which I call Robobot. This past summer, I developed the GUI using SDL (Simple DirectMedia Layer) to handle the on-screen drawing. Robobot communicates with the modules on the robot, via IPC, to handle user input and display robot state information to the user. It also uses Robomon to start modules on the robot with which it is communicating.

Role of the Simulator

The simulator has to trick all of the modules into perceiving that they are operating on a mobile robot, as opposed to a testbed computer. This means that it has to replace the physical form of the robot, including the motors and range sensors, while simultaneously providing a simulated three-dimensional environment. Said map will consist of a two-dimensional maze with walls of a set height.

Note that the simulator does not include any visual input: the robot software, specifically SVM, will not be able to see. Although this would be a desirable set of features, it would not be feasible to implement them in the time allotted for this project.

From the point of the view of the modules, the simulator simply has to replace the Mage library. Instead of communicating with physical motors through the RFLEX board via a serial connection, the library will use IPC to communicate with an active instance of the simulator module. As an option, the replacement Mage library will be able to switch into a pass-through mode, where the commands are sent directly to the serial port again.

These requirements mean that the simulator will consist of at least two parts, not all of which are integral to the simulation code. This modular format allows each part to be developed and tested independently before final integration.

Software Libraries

The core of the simulator will rely on two open source libraries to provide an external GUI display of the simulation and to give the simulation accurate physical realism. The former is provided by GTK+, a standard widget library, and the latter is provided by ODE, the Open Dynamics Engine. With ODE, I will be able to define a simple physical model of the robot that focuses on joint behavior such as that found in the motors, and the weight distribution. This model will be able to react to forces in the world over time, and respond to robot commands provided through the simulator's interface.

Project Plan

Project Plan Summary

This coding project absolutely requires a modular approach to its development. In and of itself, it will be a fairly complex piece of software, in terms of creating a physical simulation engine. Further modularity is required because it needs to integrate seamlessly with all of our existing robot code. As far as the modules are concerned, they should believe that they are running embodied in a Magellan robot, and they should receive real-time updates of realistic data.

The project has several distinct phases that encompass subsets of the tasks described in the following table. These are Phase I – Learning & Preparation, Phase II – Coding & Development, Phase III –Testing & Integration, Phase IV – Documentation & Reports.

Although the phases are linearly dependent on each other from I to IV, each of the different modules could be developed separately from initial planning to final testing, with module integration as the last and most important step. I intend to keep the development of each module in lockstep with the others, so that I don't unintentionally spend too much time working on only a single aspect of this project.

Although this project may seem fairly large in scope, I will be working on it over the entire semester. I plan to finish Phase I over Winter Break, so that I can begin coding as soon as the semester starts. Past semesters have demonstrated that I am capable of coding approximately 3000 lines a week, if necessary, on top of a full academic load.

I do not have significant classroom time commitments during the spring semester. My schedule can be dedicated almost exclusively to coding. Since I enjoy the computer work, independent of what I will be working on, I will be highly motivated to work on this project.

This section is split into three parts: a table of Task Dependencies, a complete list of Task Descriptions, and a graphical representation of the task dependencies. The critical path diagram, directly based on the table, is also included.

Note that the estimates for duration and effort in the Task Dependencies table are almost completely arbitrary. I have no good method for estimating how much coding a given task could potentially take. In addition to that, I tend to work in long bursts of coding, doing as much as 8-14 hours of work in a single sitting. This characterizes my probable weekly development schedule.

Task Dependencies

Task	Needs	Feeds	Duration	Effort	Action
A	-	G, K	3d	6h	Learn how to use ODE
B	-	C	4d	8h	Examine relevant I/O of existing modules
C	B	D, E, Q	4d	8h	Write Mage replacement wrapper library
D	C	L	1d	1h	Test false motor control
E	C	L	1d	1h	Test false range sensor input
F	-	G	1d	4h	Collect physical parameters of robot
G	A, F	K	4d	8h	Create ODE model of robot
H	-	I, P	1d	2h	Decide on a map format
I	H	J	2w	30h	Create 2-D simulation engine
J	I	K	1w	10h	Create simulation monitoring GUI
K	A, G, J	L	4d	8h	Integrate simulation engine and ODE
L	D, E, K	M, N	3d	6h	Integrate Mage wrapper with engine
M	L	O	1d	1h	Test false motor control within engine
N	L	O	1d	1h	Test false range sensor input from engine
O	K, L, M, N	Q, R	1w	10h	Integrate all engine modules
P	H	R	1w	10h	Build test worlds for simulator
Q	C, O	R	3d	6h	Give Robomon control over simulator
R	O, P	T	1w	15h	Perform extensive testing of engine
S	-	T	2m	40h	Write documentation
T	Q, R, S	U, V	3d	6h	Prepare and package for deployment
U	T	-	2w	80h	Write E90 report
V	T	-	1w	40h	Prepare E90 final presentation

For a graphical view of these task dependencies, see the path-planning diagram, with phase separation, on the last page of this proposal. Detailed descriptions of the individual tasks follow in the next section. Note that tasks U and V, the final report and presentation, are included for completeness. The body of work for the project is expected to be complete by the end of March. This allows for two weeks to write the first draft of the project report, due April 15th, and leaves the remainder of April and the beginning of May as flex time for making small changes to the project and for finalizing the report and presentation.

One of the major unforeseeable factors in any coding project is the amount of time it will actually take to get a task working, even if it is completely defined. The effort and duration estimates are padded with that expectation in mind. Hopefully the additional flex time will allow me to deal with problems as they arise.

Task Descriptions

A. Learn how to use ODE

The Open Dynamics Engine (ODE) is currently at version 0.5. All of the source code is available for download, and will compile run on both Linux and OS X, my primary development environments. There is extensive API documentation, include example implementation code, available on their website. I will have to read most of it, and attempt to implement some of the examples, in order to learn how to use the library as a part of this project.

B. Examine relevant I/O of existing modules

This is a large task simply because it will require looking over a lot of existing code, some of which is deprecated or otherwise no longer used. This is also the task for which I will most likely need Prof. Maxwell's time and assistance, because he wrote a lot of the code that is a part of these modules.

C. Write Mage replacement wrapper library

This library will consist of a set of wrapper functions that match the interface provided by Mage. Instead of sending commands to the robot's hardware through a serial connection, they will be communicated to the simulator via IPC.

D. Test false motor control

This will require a simple testing utility to confirm that the Mage wrapper is responding to commands from Nav properly.

E. Test false range sensor input

This will require a simple testing utility to inject false sensor data into Nav.

F. Collect physical parameters of robot

This experimental process will require measuring a number of the robot's physical parameters, including dimensions, mass distribution, motor responses, and range sensor limitations.

G. Create ODE model of robot

This purely physical model will consist of a cylindrical mass mounted on axial joints connected to motors. This abstract mathematical model will be implemented using ODE.

H. Decide on a map format

This is just a matter of using an existing 2-D map description, or creating one. The simplest option would be to use a black-and-white bitmap image that defines walls and open space.

I. Create 2-D simulation engine

This task is large enough to merit being a phase unto itself. At the very least, the engine needs to be capable of handling a simple maze world and the robot's motions within it. The engine also needs to be able to calculate range sensor vectors broadcast from the robot model and "reflected" off of obstacles in the virtual world. The sensor model will be extremely simplistic, and may include some noise simulation.

J. Create simulation monitoring GUI

This interface will allow the user to control the parameters of the simulation as it is running. It will also display the live sensor data and the map data as it develops.

K. Integrate simulation engine and ODE

Since my engine is fairly simple, this should not be a difficult task.

L. Integrate Mage wrapper with engine

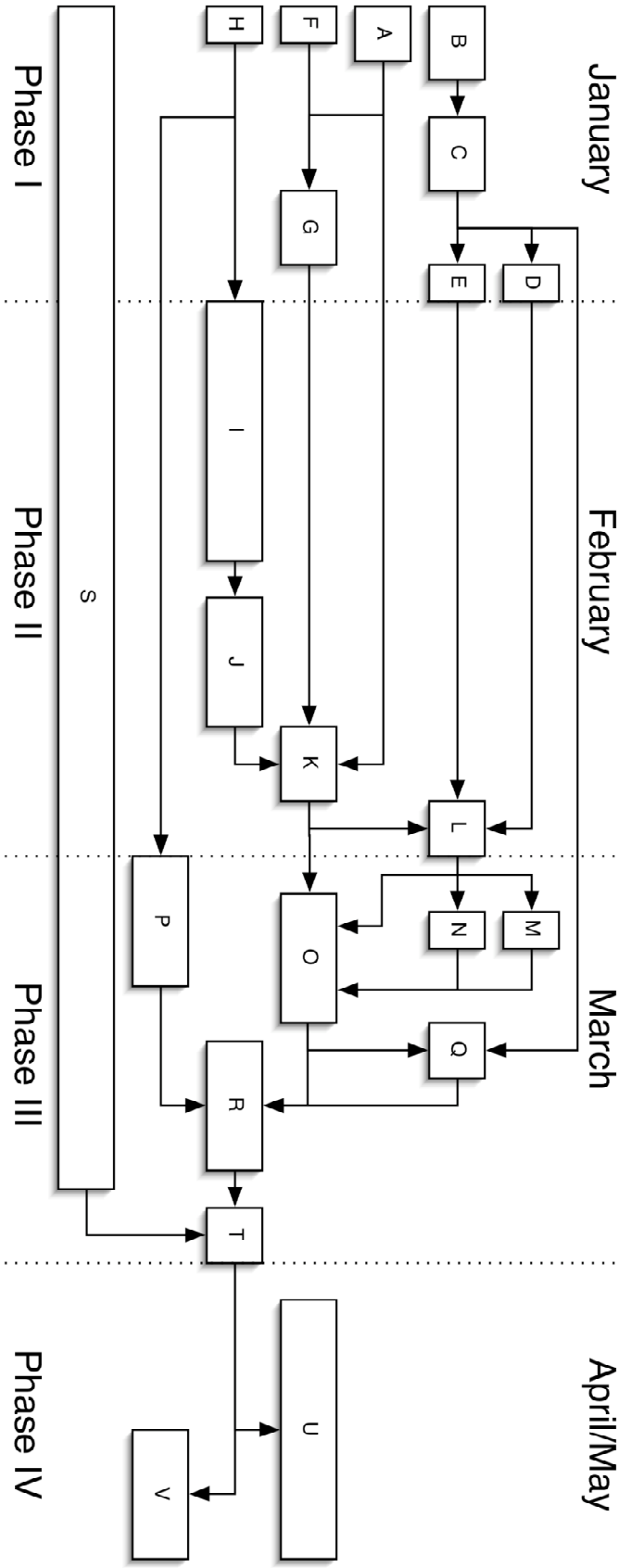
Instead of communicating with the test utilities via IPC, the range sensor and motor control messages need to be redirected to the running simulation engine. A simple task, if I initially design the modules properly.

M. Test false motor control within engine
   Confirm that task R was successful.
N. Test false range sensor input from engine
   Confirm that task R was successful.
O. Integrate all engine modules
   Assuming the individual integration tasks go well, this final integration may not be necessary, assuming that everything else works.
P. Build test worlds for simulator
   Some quality time building maze images. I may write a simple maze model generator so I can use a random maze for each test.
Q. Give Robomon control over simulator
   I am not sure if this is entirely necessary, but I think that Robomon, as the module manager, should know whether it's running on a robot or on a desktop computer. Also, it can start and stop the simulation module as necessary.
R. Perform extensive testing of engine
   Make sure that every case that I can think of works. I will possibly bring in some other people to get them to try to break the simulator.
S. Write documentation
   I plan to keep the code documentation up-to-date throughout the project, but that may not be entirely possible. The final documentation for the code (but not the project overall) will be made available as a website.
T. Prepare and package for deployment
   If I'm feeling really industrious, I'll learn how to use autoconf and automake so that the simulation package can be built on just about any system with the robot modules and other libraries. If not, I'll just use one of the good existing makefiles I have lying around, and then put all of the code into the robotics CVS repository, stored on Narya.
U. Write E90 report
V. Prepare E90 final presentation

Project Qualifications

I have had significant experience doing software development in C or C++ for large projects. Over the past 2 years, I have written well over 50,000 lines of code, as part of class assignments and summer work. Much of that work was entirely self-guided, so I know how to plan out a software development project, including design and implementation.

I also have two summers experience working with the robots directly. Although I spent most of my time writing interface modules for human teleoperation, I am very familiar with the internal workings of most of the existing robot module code.


January
February
March
April/May
Phase I
Phase II
Phase III
Phase IV
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V