Strategies for Using a Simulation in the Development of the Bremen Autonomous Wheelchair

Thomas Röfer

Bremer Institut für Sichere Systeme, TZI, FB3,
Universität Bremen, Postfach 330440, 28334 Bremen, Germany
E-mail: roefer@tzi.org

KEYWORDS

Health care, Object-oriented, Interactive simulation, Real-time simulation, Software engineering.

ABSTRACT

This paper presents a pragmatic approach to the simulation of an autonomous mobile system, the Bremen Autonomous Wheelchair. The software architecture allows to directly switch between simulation and reality. The major idea of this paper is not to try to build a perfect simulation of the target system from scratch, but to let the simulation evolve during the experiments with the real system.

INTRODUCTION

There is no disputing that simulations are useful tools in the development of autonomous mobile robotics systems. Hence, most commercial mobile robotics platforms are shipped with simulator packages that imitate the system's behavior, e.g. the *Nomad 200* or the *Pioneer*. However, these simulations are very specialized. As the mobile robot is only able to move in 2-D space, the environments that can be modeled are also restricted to two dimensions. In addition, the packages only simulate the sensors the original robot is equipped with, and it is impossible to include additional sensors. Hence, there are two major reasons to develop an own simulator: either one does not use an commercial platform for robotics experiments, and therefore has no simulator at one's disposal, or one wants to simulate more than the standard package is able to.

In the development of the Bremen Autonomous Wheelchair, both is true. On the one hand, the experimental systems are based on normal electric wheelchairs, and hence no simulation is available. On the other hand, it is not initially clear, which sensory equipment is required to reach the goal of an semi-autonomous vehicle for handicapped people. As a result, an architecture has been chosen that includes a universal robotics simulator package and adapts it to the special needs of the wheelchair application.

METHODOLOGY

The basic idea of this architecture is that it should be possible to easily switch between simulation and reality. It is not the goal to get a perfect simulation that is able to reproduce any sensor reading. Instead, the approach is to get the same *behavior* of the vehicle in the simulation as in reality. So, the simulator evolves during the development of the real system. Whenever the wheelchair's behavior simulated diverges from its real counterpart, the dissimilarity is analyzed and the simulation is expanded. This has been necessary less often than one would expect.

For example, during the implementation of the wheelchair's *basic behaviors* (Röfer 1997; Röfer and Müller 1998), it turned out that the behavior of the real system diverges from the simulated one because the simulation of the ultrasonic sensors does not include specular reflections (Lee 1996). Hence, SimRobot has been upgraded by this feature. As a result, the simulation imitates the real vehicle's behavior sufficiently close.

Thus, whenever an algorithm developed in one environment, i.e. in simulation or reality, cannot be applied to the other, the simulation has to be enhanced. To avoid to develop methods in simulation that do not work in reality, and therefore are useless, the swapping between simulation and reality should be done frequently, i.e. whenever a new idea has been successfully realized in one of both environments, it has to be applied to the other.

In the sequel, the universal robotics package *SimRobot* will be described, followed by the presentation of the adaptations to the software architecture of the wheelchair.

SIMROBOT

SimRobot is a kinematic robotics simulator that was developed in the author's group (Siems *et al.* 1994). It is written in C++ and is distributed in public domain. It consists of a portable simulation kernel and platform specific graphical user interfaces. Currently, the *X Window System*, *Microsoft Windows 3.1/95/NT* and *IBM OS/2* are supported user interfaces. During the last years, especially the user interfaces of the Windows and OS/2 versions have evolved. In fact, they are very similar. Hence, the following description will be based on these versions.

A simulation in SimRobot consists of three parts: the simulation kernel, the graphical user interface, and a *controller* that is provided by the user.

Simulation Kernel

The kernel of SimRobot models the environment, simulates sensor readings, and executes commands given by the controller. A simulation scene is described textually as a hierarchy of objects. Objects are bodies, emitters, sensors, and actuators. Some objects can contain other objects, e.g. the base joint of a robot arm contains the objects that make up the arm.

Emitters

SimRobot uses a very abstract model of measurable quantities. Instead of defining objects as lamps or color cameras that emit and measure light, it uses objects that emit intensities of particular *radiation classes* (emitters) and objects that measure these intensities (sensors). Hence, it is up to the user to define some of these abstract classes to represent real phenomena, e.g. to use the radiation classes 0, 1, and 2 to represent the three color channels red, green, and blue.

There are only two types of emitters in SimRobot: the *radial emitter* sends its radiation in all directions, whereas the *spot emitter* has a certain opening cone. In addition, an ambient intensity can be specified for each radiation class that defines the base intensity for all surfaces in a simulation scene. Hence, surfaces not reached by any of the emitters in a scene still have a sensible radiation signature.

Bodies

Currently, bodies can only be modeled as a collection of polygons. Each polygon has a radiation vector that defines its appearance—together with the radiation of emitters that reaches the surface. In addition, textures can be mapped on the surfaces to imitate e.g. wood.

Actuators

Actuators allow the user or the controller to actively influence the simulation. They can be used, e.g., to move a robot or to open doors. Each actuator can contain other objects, i.e. the objects that it moves. SimRobot provides four types of actuators: rotational joints, translational joints, objects moving in space in six degrees of freedom, and vehicles with typical car kinematics, i.e. with a driving axle and a steering axle.

Sensors

SimRobot provides a wide variety of sensors. However, only three types of information can be sensed:

- Intensities of Radiation. There are three types of cameras that allow to measure two-dimensional arrays of intensities of radiation. The *camera* object imitates normal pinhole cameras. The *facette* simulates cameras with a spherical geometry, i.e. the angle between all pixels is constant. The sensor readings can be calculated using *flat shading*, i.e. each surface has a single combination of intensities, or with different intensity signatures for every pixel. In addition, it is possible to determine shadows.

- Distances. There are several sensors that measure distances. A *whisker* can imitate the behavior of an infrared sensor. The *whisker-field* combines several whiskers to a two-dimensional array. There are *ultrasonic* sensors that measure the distance to the closest object in their measuring range. The *cameras* with flat or spherical geometries can also generate two-dimensional distance arrays that can imitate sensors currently under development or that may be interpreted as the result of a depth calculation by an image processing method, e.g. a *structure from stereo* approach (Dhond and Aggarwal 1989).
- Successful Executions. For every actuator, it can be detected whether a collision-free execution of the last command was possible. This is an information that is not available in reality. However, it allows to test algorithms that require a perfect collision detection. In addition, it allows to automatically test whether a collision occurs although the simulated robot has an obstacle avoidance mechanism.

Apart from the latter, all sensor readings can be disturbed by a selectable amount of white noise.

User Interface

The user interface of SimRobot includes an editor for writing the required scene definition files (cf. Fig. 1, above left). If such a file has been written and has been compiled error-free, the scene can be displayed as a tree of objects (cf. Fig. 1, above right). This tree is the starting point for opening further views. SimRobot can visualize any object and the readings of any sensor that are defined in a scene. Objects are displayed as wire-frames with or without hidden line removal (cf. Fig. 1, below left). The sensory data can be depicted as line graphs, column graphs, monochrome images, and color images (cf. Fig. 1, below right). In addition, depth images can be visualized as single image random dots stereograms. Any of these views and a numerical representation of the sensory data can be copied to the system's clipboard for further processing, e.g., in a spreadsheet application or a word processor. The whole window layout is stored when a scene is closed and restored when SimRobot is started again with the same scene.

Controllers

The controller implements the sense-think-act cycle; it reads the available sensors, plans the next action, and sets the actuators to the desired states. Then, SimRobot performs a simulation step and calls the controller again. Controllers are C++ classes derived from a predefined class *CONTROLLER*. Only a single function must be defined in such a controller class that is called before each simulation step. In addition, the controller can recognize keyboard and mouse events. A very powerful function is the ability to insert *views* into the scene. These are similar to sensors but in contrast to them, their value is not determined by the simulation but instead by the controller. This allows the controller to visualize, e.g., intermediate data.

SOFTWARE ARCHITECTURE OF THE WHEELCHAIR

SimRobot provides a synchronous simulation of the environment because the simulated scene “waits” until the controller completes the calculation of its next action. This is not realistic but it allows to precisely replicate a simulation run an arbitrary number of times which is quite useful during debugging a controller. Although it is possible to start developing an algorithm using such a simulation (Röfer 1995), many things have to be changed on the way to reality. Hence, for the development of the Bremen Autonomous Wheelchair, SimRobot was embedded in an asynchronous simulation architecture. It should be noted that SimRobot is used as user interface for both the simulation and the real system.

The Real Wheelchair

The first prototype of the Bremen Autonomous Wheelchair (cf. Fig. 1, right image) (Röfer 1997; Röfer and Müller 1998) has a front axle that drives the wheelchair and a back axle that is used for steering. Therefore, the wheelchair moves like a car driving backwards. It is equipped with a Pentium 100 computer, twelve bumpers, six infrared sensors, 16 ultrasonic sensors and a camera. The infrared sensors can only detect whether there is an obstacle within a radius of approximately 15 cm; however, they cannot measure the distance. Two

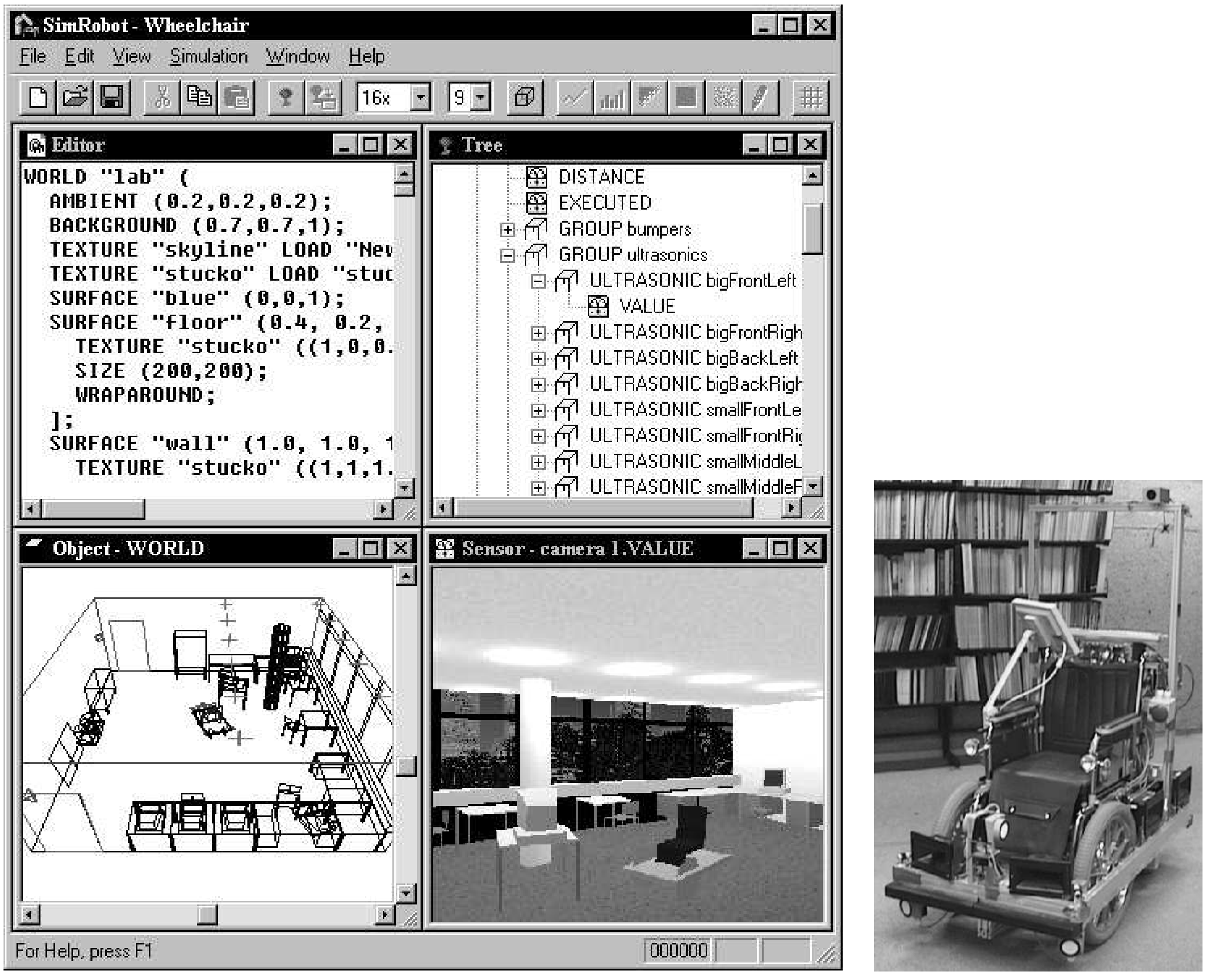

Figure 1: The user interface of SimRobot and the Bremen Autonomous Wheelchair.

different kinds of ultrasonic sensors are fitted to the wheelchair: half of the sensors have an opening-angle of 80° while the other half only measure in a range of 7°. In addition, the wheelchair can measure the rotations of its front wheels. Thus, it is able to perform dead reckoning.

The hardware of the wheelchair is controlled by five micro-controllers connected with the PC via a CAN-BUS. The PC synchronously communicates with these micro controllers in a dedicated execution thread. Hence, inside the PC, sensory data arrives asynchronously to the control program.

The Software Architecture

The software on the PC has been implemented in the object oriented language C++. The most powerful technique in object oriented programming is derivation. Thus, the software components have been structured this way. The software has been organized as a derivation hierarchy. On the lowest level, a class implements the communication with the hardware. On top of it, class by class, additional functionality is realized. The basic idea is that higher classes can change the functionality of lower ones in order to add new properties, e.g. to realize a collision detection. Thus, derived classes can add member functions and replace others.

The base class *Wheelchair* performs the communication with the micro-controllers, and hence with the hardware of the system. This communication runs asynchronously to the rest of the program. The class provides functions for setting the wheelchair's speed and steering radius. It allows to read the states of the tactile, infrared, and ultrasonic sensors. In addition, it provides the states of the two odometer counters and several functions that allow to control the pan-tilt-head. Thus, Wheelchair provides access to the complete functionality of the mobile system.

A second class, *SimChair*, links Wheelchair into the robotics simulator package. It is derived from SimRobot's *CONTROLLER* class and exists in two versions, one that uses the simulation and one that uses the real wheelchair. With the help of SimRobot, SimChair provides the same interface as *Wheelchair* but enhances it by some additional functions. Hence, any class that can be derived from Wheelchair can also be derived from SimChair. This allows to switch between simulation and reality. However, it requires that the wheelchair and its environment have to be correctly modeled in a SimRobot scene definition file. Derived classes are only allowed to use the narrow interface of SimChair to implement their functionality. As a result, e.g., the odometry calculation in a derived class is always based on the rotations of the wheelchair's front wheels—even in simulation, although SimRobot would also provide this information directly.

In the simulated version, SimRobot replaces the communication thread of Wheelchair, and thus runs asynchronously to the control program. As a result, the simulated wheelchair continues to move even if the control program performs lengthy operations—as the real wheelchair would do that does not wait for the program either.

CONCLUSION

This paper has presented a software architecture for the parallel development of a robotics system in simulation and reality. A universal simulator package has been adapted to allow the direct switching between reality and simulation. The architecture has already been used for the development of some navigation techniques (Röfer 1997; Röfer and Müller 1998). A more detailed description was given by Röfer (1998). The architecture has been proven to be a powerful tool in robotics research.

ACKNOWLEDGEMENT

The author is supported by the Deutsche Forschungsgemeinschaft through the priority program "Spatial Cognition".

REFERENCES

Dhond, U. R. and Aggarwal, J. K. (1989). Structure from stereo – a review. In *IEEE Transactions on Systems, Man, and Cybernetics*, volume 19:6, pages 1489–1510.

Lee, W. Y. (1996). *Spatial Semantic Hierarchy for a Physical Robot*. Ph.D. thesis, University of Texas at Austin.

Röfer, T. (1995). Navigation mit eindimensionalen 360° Bildern. In R. Dillmann, U. Rembold, and T. Lüth, editors, *Autonome Mobile Systeme*, volume 10 of *Informatik aktuell*, pages 193–202, Berlin, Heidelberg, New York. Springer.

Röfer, T. (1997). Controlling a wheelchair with image-based homing. In *Spatial Reasoning in Mobile Robots and Animals*, pages 66–75, Manchester University. AISB-97 Workshop.

Röfer, T. (1998). *Panoramic Image Processing and Route Navigation*. Ph.D. thesis, Universität Bremen. To appear.

Röfer, T. and Müller, R. (1998). Navigation and routemark detection of the bremen autonomous wheelchair. In *Proc. Distributed Autonomous Robotics Systems*. To appear.

Siems, U., Herwig, C., and Röfer, T. (1994). Simrobot - Ein Programm zur Simulation sensorbestückter Agenten in einer dreidimensionalen Umwelt. In B. Krieg-Brückner, G. Roth, and H. Schwegler, editors, *ZKW-Bericht*, number 1/94 in ISSN 0947-0204, Universität Bremen. Zentrum für Kognitionswissenschaften. Also: http://www.tzi.org/~simrobot.