3D mission oriented simulation

Eric Colon, Kristel Verbiest – Royal Military School
eric.colon@rma.ac.be

Abstract

In the project ViewFinder mobile robots equipped with numerous perception sensors will be integrated into a command and control network. In order to improve the development process and secure the test phase, a simulation environment is mandatory.
This paper presents the simulation needs of the project ViewFinder taking into account the robot mission planning characteristics. State of the art graphics and simulation environment are reviewed and related to the project's requirements. One of the key aspects is the integration of the simulation environment with the communication middleware that is used in the project. The evaluation and selection process is reported and the first results are presented.

Keywords: simulation, distributed control, Java3D, …

Introduction – The ViewFinder project

The project ViewFinder aims at integrating mobile robots into a command and control network to help rescue teams in case of disasters. The robots will be equipped with numerous perception sensors that will be used for the navigation of the robot itself and for the detection of dangerous products like chemical agents or toxic fumes. These data could be partially processed on board of the robot but will also be processed, stored and visualized in a remote base station. In order to increase the development process and secure the test phase, a simulation environment is absolutely necessary.
The simulation needs of the project ViewFinder have taken into account the robot mission planning characteristics. The simulation environment should be able to simulate different robots at the same time, obstacles detection, positioning and chemical sensors. It has to provide vehicle dynamics, collision detection and realistic views of the 3D environment including smoke and fire.
We have reviewed different graphics and simulation environments and related their capabilities to the project's requirements. Available features, programming language, stability, support and development tools are important evaluation criteria's but the key aspect is the integration of the simulation environment with the communication middleware that is used in the project. Two approaches are considered. The first one consists in starting from a high performance graphics engine like Ogre or Crystal Space and integrating additional functions like physics, sound and communication. The other possibility is to use an existing robot simulator like Gazebo, Microsoft Robotics Studio or COROSIM and to adapt it to suit our needs. The evaluation and comparison of these different solutions as well as the selection process is reported and the first results are presented.

The requirements in 3D simulation and visualization

The simulation needs of the project ViewFinder have to take into account the robot mission characteristics. The robot has to move into a partially unknown outdoor and indoor environment. The robot has to navigate in the environment by following a planned trajectory and avoiding unforeseen obstacles. In an outdoor environment the robot must be able to locate itself based on positioning sensors like GPS and odometry sensors or on more advanced SLAM methods. The components acquiring the sensor data and controlling the robot based on

these measurements are developed with the CoRoBA[COLO06] framework. In the real application data will be processed on board but some information could also be sent to the remote control station (position of the robot, images from cameras).

The simulation environment should consequently be able:

- to simulate different robots at the same time,
- to provide basic vehicle dynamics
- to detect collisions between robots and mobile or fixed obstacles
- to perform obstacles detection in order to simulate distance measuring sensors ,
- to simulate positioning sensors
- to simulate chemical sensors.
- to realistic views of the 3D environment including smoke and fire.
- to generate images from the scene for simulating on-board cameras
- to communicate with control components developed with the CoRoBA framework

The required simulator should consequently be able to meet these requirements.

Review of robotics simulator and 3D engines

3D Engines

Here just a selection of 3D Graphics Engines and their respective properties are represented. This is only a small subset of the abundance of available free and open source 3D engines. The engines are evaluated in function of the requirements of the project. Some evaluation criteria in choosing a graphics engine are: the used programming languages, the extensiveness of supplied features, the underlying low-level graphics API, ease of use, extendibility, support and popularity, extendibility, stability... Furthermore, the engines are preferably open source, free and cross-platform.

OGRE

OGRE [MILN04] (Object-Oriented Graphics Rendering Engine) is a free, open source 3D Graphics Rendering Engine, not a gaming engine and therefore does not provide built in physics, sound support, network support, etc. Nevertheless these can be plugged in if needed. It has an OO-oriented design and the programming language used is C++. It runs both on Direct3D and OpenGL. The rendering is based on scene graphs. OGRE is primarily used as a rendering engine in the development of games. Furthermore it is cross-platform, stable, provides advanced rendering features (latest technology) and great graphics, and has a good performance, good documentation and support. Nevertheless it has a steep learning curve and is not suited for beginners.
Applications that use OGRE are for instance ‘Arid Ocean’, which is a lifelike simulation of an underwater world, see figure 1, and Live Interior 3D, which provides tools for designing 3D interiors, see figure 2.

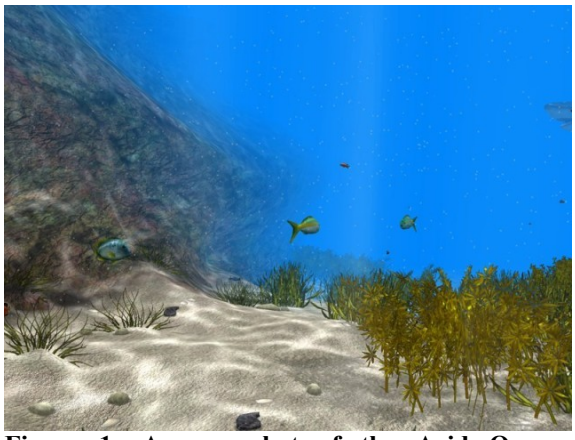

Figure 1: A screenshot of the Arid Ocean application developed with OGRE as graphics engine.

Figure 2: A screenshot of the Live Interior application developed with OGRE as graphics engine.

Panda3D

The Panda3D[GOSL04] (Platform Agnostic Networked Display Architecture) Engine is developed jointly by Disney and Carnegie Mellon University's Entertainment Technology Center. The Panda3D Engine provides 3D rendering and game development. Panda3D is an all in one engine as it provides support for: sound, input, animations, behaviours, networking, etc. It runs both on OpenGL and Direct3D and offers support for both the Windows and Linux operating systems. The library is written in C++ with a set of Python bindings. A rapid development cycle is provided due to programming in Python. Moreover it is free, open source, stable, flexible, powerful and has a good performance. Other favourable qualities are the short learning curve and excellent support and documentation (tutorials, active forum, and API reference).

Panda3D only supports the basic features and not the more advanced ones, like high end commercial engines. There are gaps in the manual for rarely used features like networking, but one can turn to the forum for help. It can also be used in C++, but this is not very well documented and supported. Disney's massive multiplayer online role playing game (MMORPG) Toontown Online was built using Panda3D, see figure 3.

Crystal Space

Crystal Space[TYBEG07] is a free, open source 3D engine that finds its applications mostly in the field of modelling and simulation, games and real-time 3D graphics. Crystal Space offers (almost) everything that is needed for a game or other 3D graphics applications: graphics, sound, IO, animation, dynamics/collision detection, terrain, GUI support… and is mostly geared toward the larger projects. It is cross-platform, has a modular design and uses C++ as programming language. Crystal Space provides many advanced features, plug-ins for almost anything, a good performance, and has a dedicated developer base, a helpful community and good documentation (tutorials, manuals, demos).

Nevertheless Crystal Space can be complex for the less experienced programmers and has a very steep learning curve. At first, it can be very intimidating and difficult to use, as it is hard to set up and hard to master. One has to face the initial challenge of learning the API.

For an example of the use of Crystal Space, see figure 4, which shows a screenshot of the famous commercial game Keepsake.

Figure 3: Disney's online role playing game Toontown Online, built using Panda3D.

Figure 4: A screenshot of the famous commercial game Keepsake, made with Crystal Space.

Simulators

Here a selection of simulators is discussed: Marilou Robotics Studio, Microsoft Robotics Studio and the Player/Stage project.

Marilou Robotics Studio

Marilou Robotics Studio[MARI07] is a 3D modelling and simulation environment for mobile robots operating in a real-world conditions that respect the law of physics (physical simulation using Open Dynamics Engine, ODE) and runs on top of DirectX. Real-time, interactive and multi-robot simulation is offered. Robot programming can be done using various programming languages: C/C++, VB#, C#, J# CLI. Libraries with embedded robotic components are available: distance sensors (infrared, ultrasonic, laser), motors, odometers, GPS, cameras…

It is easy to use and has an intuitive GUI, figure 5 shows a screenshot of the GUI. It is an all-round simulator with lots of features and possibilities.

Marilou robotics studio is not open source, nor free and only runs on the Windows operating system.

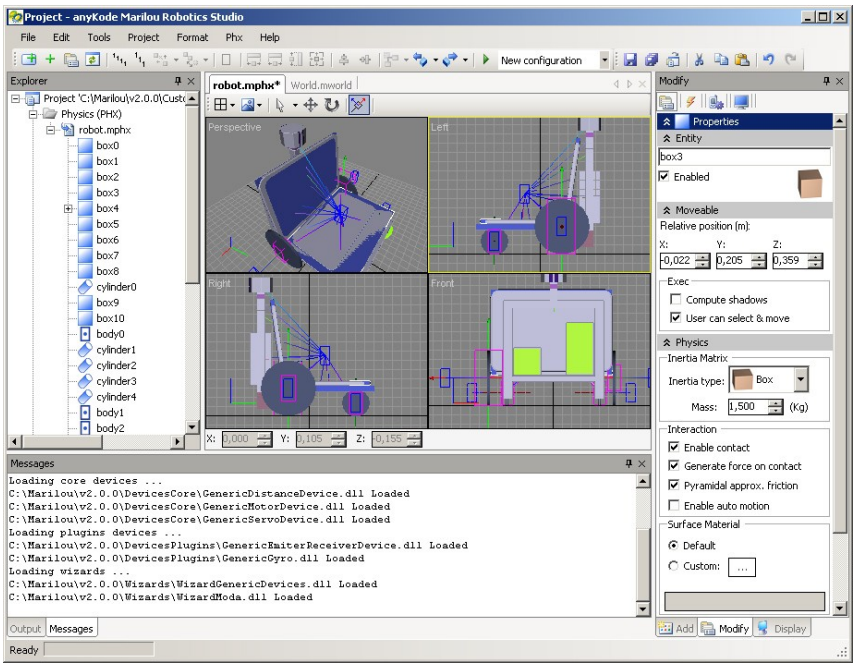

Figure 5: A screenshot of the GUI of Marilou Robotics Studio

Microsoft Robotics Studio

Microsoft Robotics Studio[MICR07] is a Windows-based environment to create robotic applications for a variety of hardware platforms. According to Microsoft's description, Microsoft Robotics Studio delivers three areas of software:

End-to-end robotics development platform. Microsoft Robotics Studio includes a visual programming tool, making it easy to create and debug robot applications. Robotics Studio enables developers to generate modular services for hardware and software, allowing users to interact with robots through Web-based or Windows-based interfaces.
Developers can also simulate robotic applications using realistic 3-D models; Microsoft has licensed the PhysX engine from AGEIA, a pioneer in hardware-accelerated physics, enabling real-world physics simulations with robot models. The PhysX simulations can also be accelerated using AGEIA hardware. Figure 6 and 7 show screenshots of the simulation environment, displaying the rendered and the physical entities respectively.

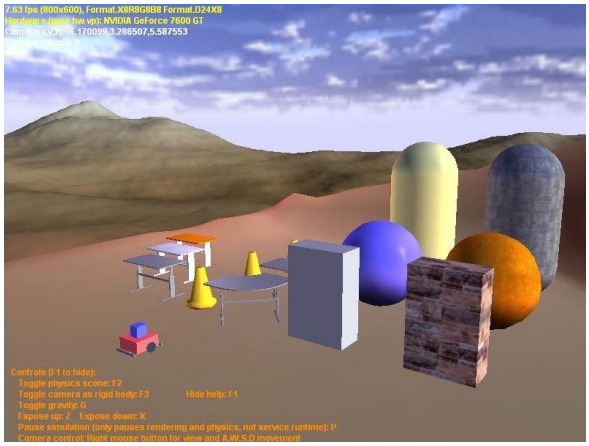

Figure 6: A screenshot of the Microsoft Robotics Simulation tool.

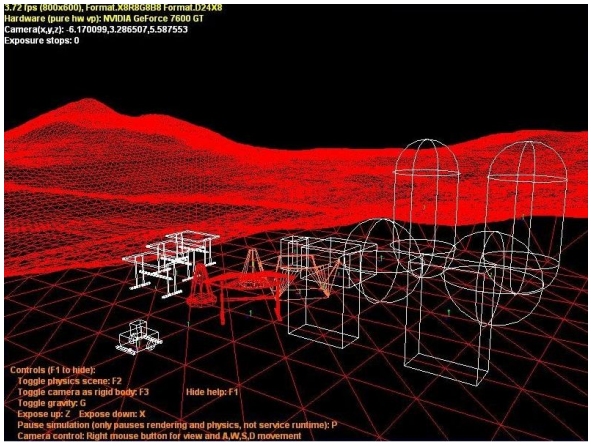

Figure 7: A screenshot of the Microsoft Robotics Simulation tool showing the physics of the entities in figure 4.

Lightweight services-oriented runtime. Using a .NET-based concurrency library, asynchronous application development is made simple. The services-oriented, message-based architecture makes it simple to access the state of a robot's sensors and actuators with a Web browser or Windows-based application, and its composable model enables the building of high-level functions using simple components and providing for reusability of code modules as well as better reliability and replaceability.

Scalable, extensible platform. The Microsoft Robotics Studio programming model can be applied for a variety of robot hardware platforms, enabling users to transfer their learning skills across platforms. The programming interfaces can be used to develop applications for robots using 8, 16 or 32-bit processors, either single or multi-core.
Third parties can also extend the functionality of the platform by providing additional libraries and services. Both remote (PC-based) and autonomous (robot-based) execution scenarios can be developed. With Microsoft Robotics Studio, robotics applications can be developed using a selection of programming languages, including those in Microsoft Visual Studio and Microsoft Visual Studio Express languages (Visual C# and VB.NET), as well as Jscript and Microsoft Iron Python. Third-party languages that support the Microsoft Robotics Studio services-based architecture are also supported.

The Player /Stage Project

Player/Stage[GERK03] is a widely used middleware in the robotics community. Player is a robot server that provides network transparent robot control. The project provides two multi-robot simulators: *Stage* and *Gazebo*. Since Stage and Gazebo are both Player-compatible, client programs written using one simulator can usually be run on the other with little or no modification. The key difference between these two simulators is that whereas Stage is designed to simulate a very large robot population with low fidelity, Gazebo is designed to simulate a small population with high fidelity.
Some of the key features of the project are: platform, language, and transport protocol independence; enhanced scalability; open source; usage of standardized communication protocols and high modularity. The software runs on Linux, Solaris and BSD.

Player

Player is a network server for controlling multiple robots. It provides an abstract interface to the sensors and actuators on a real robot which is connected to the server using the TCP/IP protocol. An important feature of the framework is that Player is designed to work transparently both with real robots or either of the simulation environments. Modelling of robots and the environment is done using XML files with predefined environment elements and robots. Apart from the available robot models, new robots can be defined using the C programming language

Stage

Stage is a scalable multiple robot simulator; it simulates a population of mobile robots moving in and sensing a two-dimensional bitmapped environment, controlled through Player, see figure 8 for a screenshot. Various sensor models are provided, including sonar, scanning laser rangefinder, pan-tilt-zoom camera with colour blob detection and odometry. Stage devices present a standard Player interface so few or no changes are required to move between simulation and hardware.

Gazebo

Gazebo is a multi-robot simulator for outdoor environments. It is capable of simulating a population of robots, sensors and objects in a three-dimensional world, see figure 9 for a

screenshot. It generates both realistic sensor feedback and physically plausible interactions between objects (it includes an accurate simulation of rigid-body physics). Gazebo is based on the Open Dynamics Engine (ODE) which is an open-source rigid body dynamics library. Scenes are visualised in 3D by means of the OpenGL library.

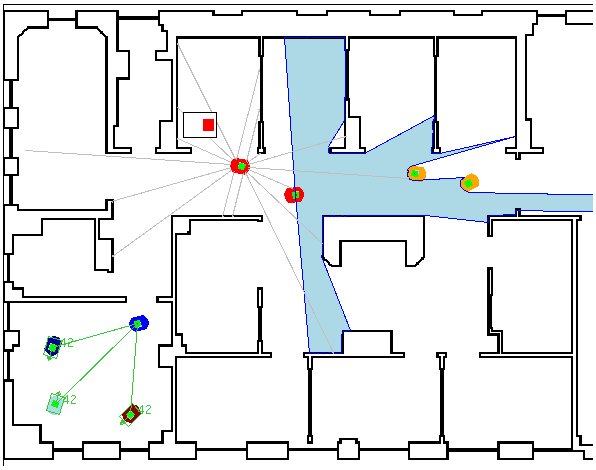

Figure 8: A screenshot of Stage, showing the 2D-simulation of multiple robots.

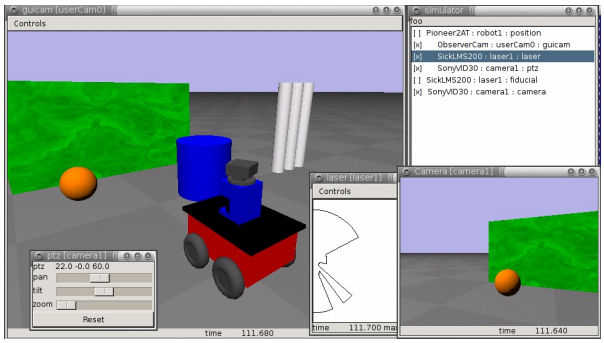

Figure 9: A screenshot of Gazebo, showing a couple of control panels.

COROSIM Description

COROSIM is a Java based 3D simulator for mobile robots. This application is able to simulate realistic motion of different simulate realistic motion of different wheeled mobile robots including dynamic behaviour and collision detection. Typical sensors are also available in order to develop intelligent navigation applications. As this simulator provides CORBA interfaces for every active object, applications can be written in any language supporting this standard.

Simulator Overview

For the user, the visible output of the simulator is a synthetic image. Actually, it is not only an image but it is also a model that is built with algorithms based on physical laws and using well defined data structures. The simulator provides the following functionalities:

- Real-time simulation of multiple robots concurrently
- 3D real-time visualization of the simulation
- User interaction through a GUI
- Dynamic control of mobile robots
- Detection of and appropriate reaction to collisions between mobile and fixed objects
- Simulation of position and distance sensors
- CORBA interfaces

The simulation process is divided in two main steps: the modelling of 3D scenes and robots by a human and the utilization of the modelled objects in the simulator. These two steps are explained in the following sections and illustrated by the diagram of Figure 10.

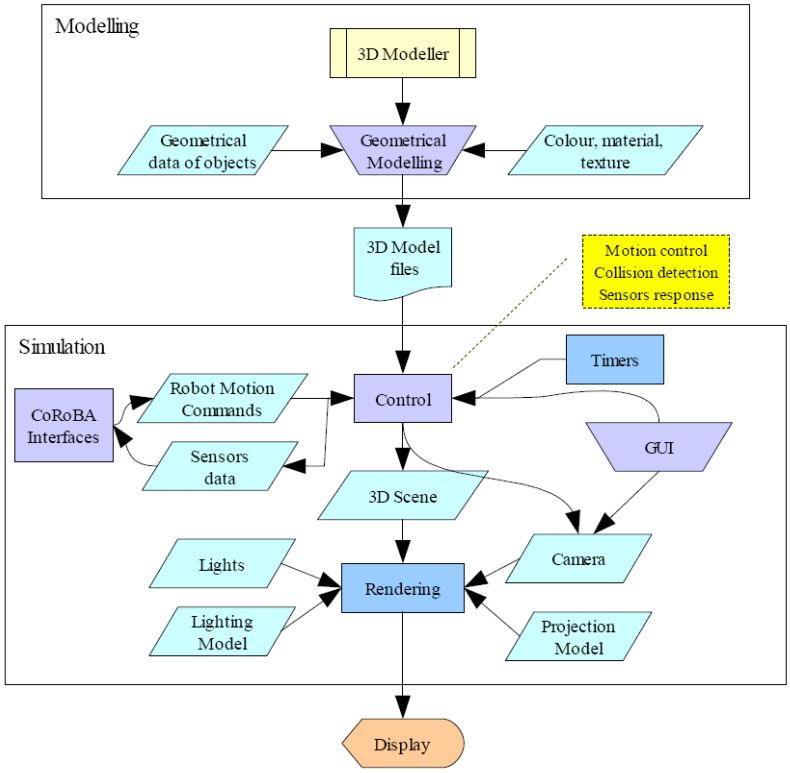

Figure 10: The simulation process.

Starting from a real or a hypothetical robot, the creator uses a 3D drawing program to generate a virtual model. Other information like colours, material and texture can be applied to the objects to improve the realism. Real or imaginary environments (terrain and obstacles) are created separately from the robots. Then the model is exported in the VRML (Virtual Reality Modelling Language) format. The VRML file is then imported by Java3D into a Java3D scene graph and inserted in the global 3D scene. The process flow (control-rendering-display) represented in Figure 10 continuously runs until the application finishes. The control process updates the Scene (section Scene Graph), controls the motion of the robots (section Robot Models), performs the collision detection and response (see section Handling Collisions) and finally computes the output of position and distance sensors (see section Sensors). The Control also receives motion commands for the robots and sends sensors' data via the CoRoBA interfaces (see section CoRoBa). It can also control the camera motion in automatic tracking mode. The GUI, see figure 11, lets the user chose the camera mode and position and gives the possibility to position the robot in the virtual world. The execution of the control process is triggered by timer events. As each robot and sensor is represented by separate objects, the events are propagated to all of them. This means that all motion and measurements are synchronized.

Once all transformations of the 3D scene have been performed, the scene is rendered by the Java3D rendering engine. The rendering engine of Java3D can use the DirectX or OpenGL libraries.

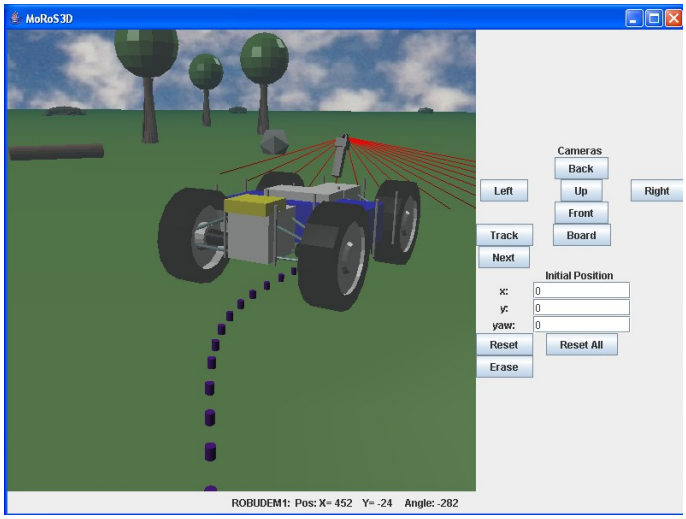

Figure 11: GUI of COROSIM , showing the robudem robot.

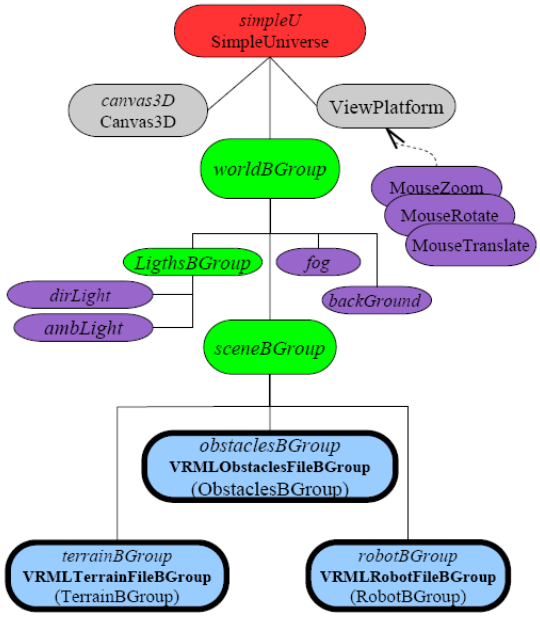

Figure 12: Scene graph.

Scene Graph

Many free and open-source toolkits are available for building 3D applications. However, most of them focus on visual aspects and few offer high level facilities for managing scenes. This is one reason justifying the use of Java3D for the development of the simulator. Java3D is a full-featured API for interactive 3D graphics. It is based on a high-level scene graph programming model that describes the scene, Java3D managing the display of it. Scene graphs are treelike data structures used to store, organize and render 3D scene information. They are made up of objects called nodes, which represent objects to be displayed, aspects of the virtual world or groups of nodes. By default, each object in a Java3D scene is initially stationary and remains at its starting location unless code specifies otherwise. The scene graph of the simulator can be seen in figure 12.

Robot Models

The geometry of robots is determined by their shape and dimensions. The 3D models have been drawn with a 3D modelling application (Wings3D) and exported in the VRML format. See figure 11 for a model of the robudem robot.
The position, orientation, speed and steering speed of the robot are updated at equally spaced intervals.

Handling Collisions

One of the basic requirements of the simulator is to provide collision detection to detect when the control algorithm fails and to provide adapted response.
A collision detection algorithm exploiting Java3D behaviours has been implemented. For any realistic environment and even if simplified shapes are used, the collision detection needs a lot of mathematical operations, therefore it is important that collision detection be very efficient. A method has to be applied to speed up the computation. For instance, bounding volumes can be used to reject non intersecting objects. However one cannot rely on bounding volumes alone if the objects are complex shapes; and also for working out the collision response the points of impact need to be known. The method used in the simulator consists in replacing mobile robots by a good approximation and checking for collision with the real geometry of other objects. Once a collision has been detected, one can compute the appropriate reaction by using physical laws. In COROSIM , the robot will be stopped in case of a collision.

Sensors

Two kinds of sensors are necessary for developing intelligent control applications in mobile robotics: position and environment perception sensors. Global position sensors can be easily implemented within the simulator because we perfectly know the position and orientation of the robot and of all its components. Distance sensors are mandatory for seeing what stands around the robot. Three models of such sensors have been implemented in the simulator, namely laser, infra-red and ultrasonic sensors. To implement the measurement process we have used Java3D's picking routines. The idea is to cast a ray into the space around the robot. This ray has a length equal to the maximum distance the sensor can measure. See figures 13 and 14 for a view of the simulated laser and ultrasonic sensor.

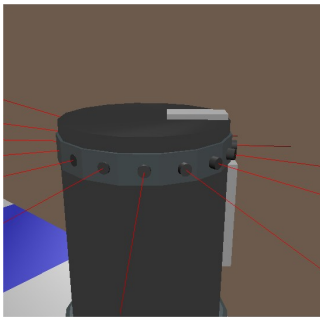

Figure 13: Simulated laser sensor.

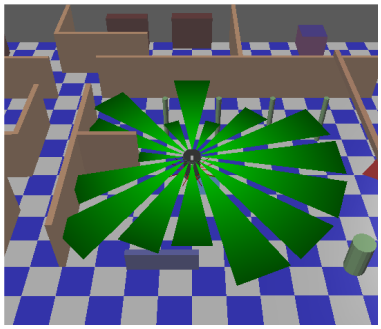

Figure 14: Simulated ultrasonic sensor.

CoRoBa

COROSIM integrates seamlessly with the control framework. CoRoBA is a solution package for developing distribution applications that uses components with standardized interfaces and communication mechanisms. Components are divided in Actuators, Processors and Sensors.

The utilization philosophy is to develop and tune control algorithms in simulation and to simply replace simulated by real components once satisfying results have been reached, no further modification of the Processor components being required. In Figure 15, the concept of integrating COROSIM in the CoRoBA framework is shown. Sensor and Actuator components developed with CoRoBA can be seen as interface components that have to be specific for the simulator or the hardware they are linked to.

The block named “Intelligent Control” on top of figure contains Processors. This part does not care if real or simulated hardware is used. The Processor components are the key-stone of the control architecture and exhibit the largest potential of reuse between applications involving different robots while Sensors and Actuators, that serve as interfaces or translators between the software and external modules, are specific to these devices.

The middle block corresponds to interface components that make the link between the Processors and the simulated world. Sensor and Actuator components implement the same interfaces as those implemented by components linked to physical systems, allowing to instantaneously switch between simulation and reality.

The last block represents the simulator. It contains models of the physical elements: robots, sensors and the environment. The robot simulator is responsible for the realistic motion of the robot and takes care of the collision with fixed and moving obstacles like other robots. It receives motion commands from Actuator components.

The simulated sensors produce measurement data that are injected in the application control loop by the Sensor components. The data is forwarded to Processor components where they are exploited to finally produce motion commands that are sent to the Actuator Components. These Actuator Components adapt and send this information to the robot objects. The sensors affect the vehicles motion through Intelligent Control and vehicles motion affect sensors through the Simulator taking into account the model of the environment.

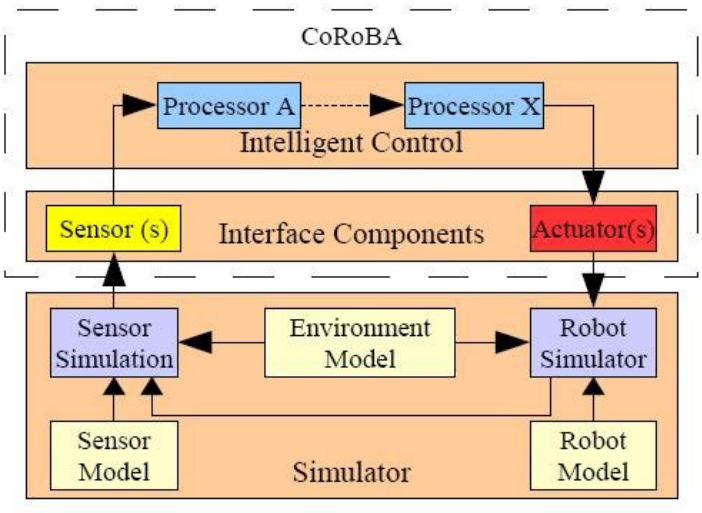

Figure 15: Simulator and CoRoBa integration.

COROSIM practical use

Goal Navigation with the Robudem

The purpose of this application is to let the Robudem move autonomously from a given position to succession of goals in an obstacle free environment. The components involved in this application and the transferred data are shown in figure 16. The Goal Controller uses a Fuzzy Inference System.

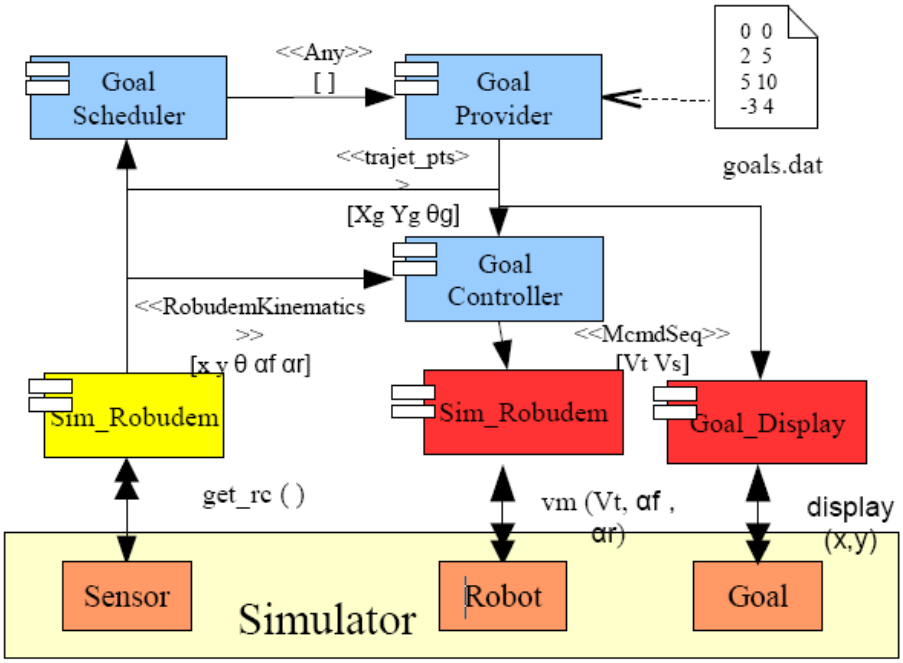

Figure 16: Component's network.

During the execution of the application, the following operations are executed:

- At initialization, the *Goal_Provider* reads a list of goals from a file (goals.dat).
- When the components are started, the first goal position [Xg Yg θg] is sent to the *Goal_Controller* and to the *Goal_Scheduler*.

- These components also receive the global position of the robot [x y θ αf αr] from the *Sim_Robudem* Sensor component).
- The *Goal_Controller* uses this information to produce steering and driving commands [Vt Vs] in order to reach the goal.
- These commands are received by the *Sim_Robudem* Actuator that adapts them to the controlled robot.
- The *Goal_Scheduler* compares the goal position received from the *Goal_Provider* with the instantaneous position of the robot. When the robot is sufficiently close to the goal, the *Goal_Scheduler* sends an event to the *Goal_Provider* to inform it that it has to provide the next goal.
- A new goal is sent to the *Goal_Controller* and *Goal_Scheduler*.
- These operations are repeated until the last goal is reached.

Two step of a navigation sequence can be seen in figure 17.

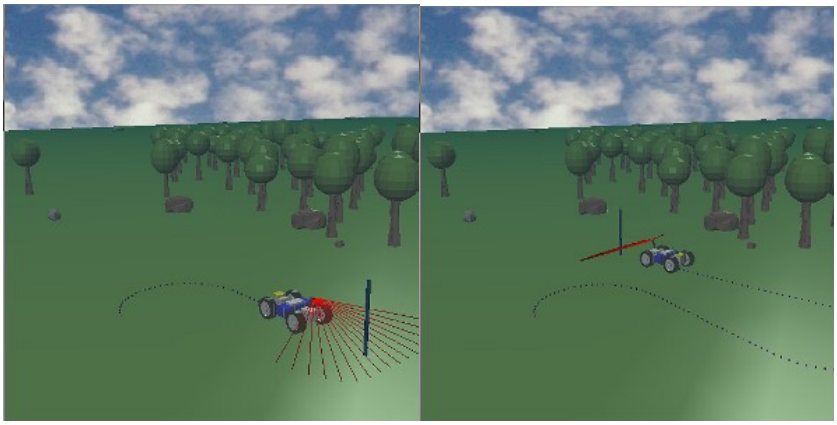

Figure 17: Two steps of a navigation sequence.

COROSIM 's Advantages

The requirements above list the properties the simulation environment must fulfil. By creating a simulator of our own, we can customize it, to more fit these needs of the ViewFinder project. We can assemble the different parts (rendering engine, physics engine, sound support networking,…) at will, resulting in a custom made simulation environment.
Existing simulators try to provide a complete package, with an as wide range of applications as possible. They are more general and a lot of design decisions and choices in software packages have already been made. Therefore they will be more difficult to alter and optimize. The COROSIM simulator is more easily adaptable to the project's specific requirements.
To be able to integrate with the CoRoBa framework, the simulator needs to have CORBA interfaces. CORBA (Common Object Request Broker Architecture) is OMG's (Object Management Group) vendor-independent architecture and infrastructure that computer applications use to work together over networks. Using the standard protocol IIOP, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network. COROSIM 's CORBA interfaces allow for communication with the control components developed with the CoRoBa framework.
The COROSIM simulator is developed using Java3D. Java3D is one of the few open-source toolkits that offer high level facilities for managing scene, instead of just focusing on the visual aspects. Because it is based on Java3D it is therefore platform independent.
One can easily change the environment, consisting out of terrain and obstacles, as it is passed as command line parameters.

The simulator can also be used independently of the framework and control applications can be written in any language supporting CORBA interfaces. Keeping the control algorithms out of the simulator has the advantage that an application developer does not need to deal with Java3D programming.

Further developments

Further improvements on the currently used simulator COROSIM can be made. Instead of applying the textures on the 3D model directly in Java3D, this can be done beforehand and separate of Java3D by using a 3D modeller and then loading the model in Java3D. Also multiple camera simulation and the simulation of smoke, fire, fog... needs to be added to the simulator. Other areas of possible improvements are a more realistical physics simulation (inertia, friction, shock, appropriate reaction to collision...) and an improved Java rendering engine like jME. Some of these improvements are discussed in more detail below.

Loading VRML models with textures

Java3D allows for a variety of formats of 3D models to be imported, like VRML, OpenFLT, Milkshape3D, 3DS, OBJ, VTK and many more. All that is needed is an appropriate loader. The 3D models used in the COROSIM simulator can be created with a variety of 3D modelling programs. In COROSIM the VRML (Virtual Reality Modelling Language) format is used to import 3D models. So any 3D modelling program that can export to VRML will do fine. VMRL is a language that has been developed for describing 3D virtual environments for Web based applications. In a VRML file other than just geometrical information can be stored; also textures, colour and material properties can be embedded. This way the 3D models used in the simulator can be created separately from the simulator, which will make the modelling of the robots, the terrains, and environments and obstacles easier as the properties of the models are in the description of the file and don't have to be programmed in Java3D. This will also allow easier reuse of the developed 3D models in other applications.

Camera Simulation

In the simulation environment there is a need to be able to simulate multiple cameras in the scene. In Java3D one can simulate multiple cameras, with each their viewpoint, by adding multiple ViewPlatforms to the View Branch Graph of the Java3D scene graph structure. Instead of displaying the camera images on screen they can be rendered off screen and sent to a buffer. From there on the images can be used for further processing.

Simulation of Smoke, Fire...

For a realistical simulation of a crisis situation, like for instance an explosion or a chemical reaction, there is a need for the simulation of smoke and fire. These special effects can be accomplished by the introduction of a particle system. A particle system is a technique to model 'fuzzy' objects, such as fire, explosions, smoke, clouds, fog, dust, sparks, etc. These 'fuzzy' objects do not have smooth, well-defined, and shiny surfaces; instead their surfaces are irregular, complex, and, ill defined. In [REEV83] a particle system is described to model a fuzzy object as a cloud of primitive particles or points in 3 dimensions. The resulting model is able to represent motion, changes of form, and dynamics, that are not possible with classical surface-based representations. How a particle changes or moves is base on a controlled stochastic process giving it a natural look. How particles evolve in a particle system is called the particle life cycle.
In [JACO05] such a particle system is created for Java3D. This particle system will be incorporated in the COROSIM . An example of the 3D simulation of smoke and fire can be seen in figures 8 and 9 respectively.

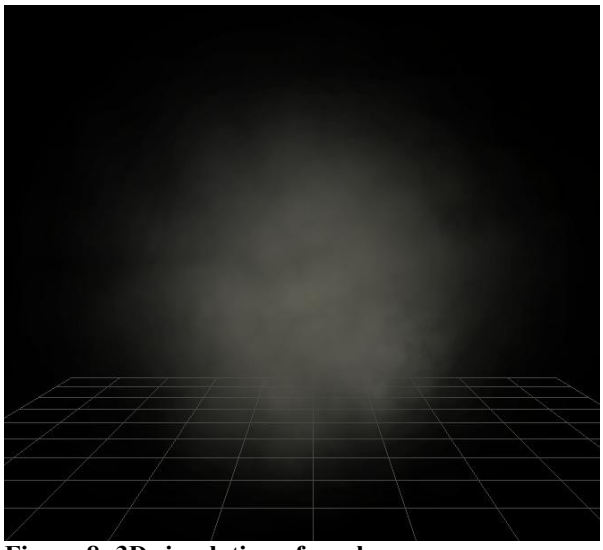

Figure 8: 3D simulation of smoke.

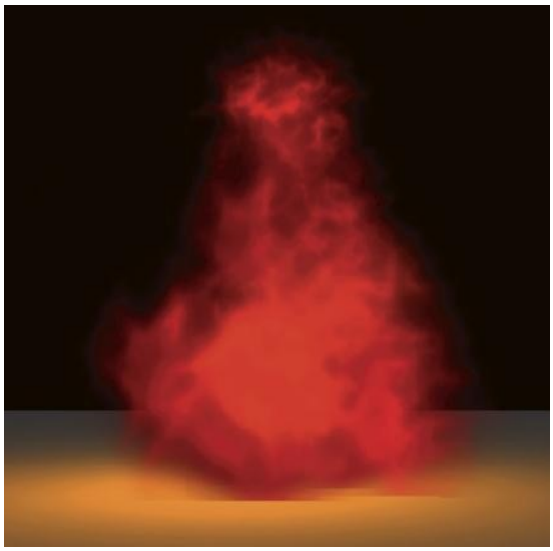

Figure 9: 3D simulation of fire.

Conclusions

COROSIM has been started as a convenient tool for testing components developed with the CoRoBA framework. For this reason the focus wad not on 3D nice graphics but on usability and communication. As it has revealed to be a valuable tool in the development of control architectures, it has been decided to be extended to meet the requirements of the ViewFinder project. Textures, smoke and fire, 3D terrain will be progressively added to COROSIMD3D to obtain a powerful simulation environment.

Acknowledgement

The development described in this paper are supported by the **View-Finder FP6 IST 045541 Project.**

References

[COLO06] Eric Colon, Hichem Shali, Yvan Baudoin,CoRoBa, a multi mobile robot control and simulation framework, *Special Issue on "Software Development and Integration in Robotics" of the International Journal on Advanced Robotics*, pp 73-78,Volume 3, Number 1, March 2006.

[GERK03] Brian P. Gerkey, Richard T. Vaughan, Andrew Howard, The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems, In *Proceedings of the International Conference on Advanced Robotics (ICAR 2003),* pages 317-323, Coimbra, Portugal, June 30 - July 3, 2003
"The Player/Stage Project"
URL: http://playerstage.sourceforge.net/

[GOSL04] Goslin, M. and Mine, M.R., The Panda3D graphics engine, Computer Volume 37, Issue 10, Oct. 2004 Page(s): 112 – 114.ISSN: 0018-9162
"Panda3D"
URL: http://panda3d.org/

[JACO05] Mike Jacobs, Star Trek Technology for Java3D: Building a Particle System for Java3D, In Java Developer's Journal Vol.:10, Iss.: 6, 2005

[MARI07] “Marilou Robotics Studio”, 2007
URL: http://www.marilou-roboticsstudio.com

[MICR07] “Microsoft Robotics Studio”, 2007
URL: http://www.microsoft.com/robotics

[MILN04] I. Milne and G. W. A. Rowe, OGRE: Three-dimensional program visualization for novice programmers, *Education and Information Technologies*, 9(3) (2004), Kluwer Academic Publishers, pp.219-237. ISSN: 1360 2357
“OGRE Object Oriented Graphics Engine”
URL: http://www.ogre3d.org/

[REEV83] William T. Reeves, Particle Systems: A Technique for Modelling a Class of Fuzzy Objects, In Computer Graphics Volume 17, nr. 3, 1983

[TYBEG07] Jorrit Tyberghein, “Crystal Space ", 2007
URL : http://www.crystalspace3d.org/