Mobile robot simulators and their application to hazardous and challenging environments

L. Nomdedeu*, J. Sales*, E. Cervera*, J. Alemany*,
R. Sebastia†, K. McAllister†

Abstract

The Guardians concept requires the design, implementation and integration of different types of components. Guardians uses multiple existing robot platforms carrying intelligent sensors for navigating and surveying in firefighting scenarios. Program code developed should be compatible for all platforms. In addition, the use of a simulator as a previous step to the experiments may be useful in order to evaluate algorithms and techniques. Many robot programming tools – notably open source – have been designed in recent years to provide an abstraction of the physical robot. In Guardians, such a common robot programming environment for the different mobile platforms is a must. In this paper, we validate the use of this simulation and development environment. An overview of the software and its elements is presented. We will describe the complete steps for installing, configuring, and running the software. The whole task of setting up a new scenario will be thoroughly described, including the different parts (world, models, configuration). To conclude, we will present some final remarks and lines of ongoing research.

1 Introduction

In the Guardians project, multiple robot platforms and sensors are used, so it requires concept requires the design, implementation and integration of different types of components. Those robot platforms and sensors are used for navigating and surveying in firefighting scenarios. Small robots will be used for real experimentation in lower scale models. Solutions will then be implemented in medium sized platforms for testing first in real scale laboratory environments, then in user scenarios. The need for a common development environment arises from the use of different platforms. Program code developed for the small platforms should be compatible, with minor adjustments, with the bigger size platforms. In addition, the use of a simulator as a previous step to the experiments may be useful in order to evaluate algorithms and techniques, especially if a group of robots is used, as is the case in swarm robotics. Finally, computer simulation is a pedagogically proven technique for training and transfer of skills, which may be exploited in the development of human robot interfaces. The list of choices is wide too in the field of development environments for autonomous

*Robotic Intelligence Lab, Jaume-I University, Castelló (SPAIN)
†Robotnik Automation, SL, Valencia (SPAIN)
mobile robots. Nearly every robot manufacturer has developed his own environment, usually restricting it to a particular programming language. Many robot programming tools—notably open source—have been designed in recent years to provide an abstraction of the physical robot, thus allowing the reuse of robot software, and making robot development faster, easier and more efficient. In Guardians, such a common robot programming environment for the different mobile platforms is a must. The source code needs to be compatible, with minor adjustments, among all the robot platforms used in the project.

2 Robotic Development Environments and Simulators

Among the components that the Guardians projects need, it is necessary to describe the robot platforms. Guardians uses multiple existing robot platforms carrying intelligent sensors for navigating and surveying in firefighting scenarios. The mobile platforms used by Guardians are composed by:

1. small platforms produced by SME partner K-Team, namely the Khepera-III robot.

2. medium-sized commercial platforms, namely Super Scouts, Pioneer, and Erratic platforms, owned by research partners.

3. the Rescuer tracked platform developed by SME partner Robotnik

Small robots will be used for real experimentation in lower scale models. Solutions will then be implemented in medium sized platforms for testing first in real scale laboratory environments, then in user scenarios.

The need for a common development environment arises from the use of different platforms. Program code developed for the small platforms should be compatible, with minor adjustments, with the bigger size platforms.

In addition, the use of a simulator as a previous step to the experiments may be useful in order to evaluate algorithms and techniques, specially if a group of robots is used, as is the case in swarm robotics. Finally, computer simulation is a pedagogically proven technique for training and transfer of skills, which may be exploited in the development of human robot interfaces.

In a recent survey [CMBG07], Craighead et al. conclude that nowadays there is a wide choice of commercial and open source available simulators, allowing resources to be focused on research, instead of building a new simulator from scratch.

The list of choices is wide too in the field of development environments for autonomous mobile robots. Nearly every robot manufacturer has developed his own environment, usually restricting it to a particular programming language.

Many robot programming tools—notably open source—have been designed in recent years to provide an abstraction of the physical robot, thus allowing the reuse of robot software, and making robot development faster, easier and more efficient.

In Guardians, such a common robot programming environment for the different mobile platforms is a must. The source code needs to be compatible, with minor adjustments, among all the robot platforms used in the project.
A study on available development environments has been published recently [KS07], aiming to a systematic evaluation of nine open source, freely available environments for mobile robots. Such environments have come to play an increasing important role in robotics research in general, thus a systematic evaluation is carried out, based on features, usability and impact issues.

Based on the above-mentioned surveys, and previous experience of partners, the Player / Stage / Gazebo platform [VG07] has been chosen for simulation and robot development in Guardians. This platform is already available for some platforms used in the project. Since it is open source and extensible, appropriate drivers for the rest of the platforms will be developed by partners K-Team and Robotnik.

The Player / Stage / Gazebo project is an open source platform that provides either a 2D and 3D environment for robot simulation. The former (Stage) is well suited for simulations of a huge number of robots like in swarms, while the latter (Gazebo) provides basic physics simulation for a reduced number of robots.

In [CMBG07], Player / Stage scores “high” in terms of ease of development, “medium” in physical fidelity and cost, and “low” in functional fidelity (the authors state that those ratings are subjective in terms of the best technology available to date). As for a comparison among development environments, in [KS07], Player / Stage achieves the maximum score (70 / 100) among the nine compared environments. That score consists of the sum of feature, usability and impact scores.

In the following sections, we validate the use of this simulation and development environment. In section 3, an overview of the software and its elements is presented. Then, section 4 describes the complete steps for installing, configuring, and running the software. The whole task of setting up a new scenario is thoroughly described in section 5, including the different parts (world, models, configuration). Finally, section 6 presents some final remarks and lines of ongoing research.

3 The Player / Stage / Gazebo tools

3.1 The Player robot device interface

Player is a control layer, a middleware, that provides a common interface to heterogenous robot platforms and sensors. It is an open source project supported by many community developers interested in robotics. By using Player you are allowed to control either a real robot, or a simulated one provided by the Stage simulation tool. Moreover, you will achieve a fast and robust development, being able to test and to deploy your code written in any of the supported programing languages with an incredible simplicity.

3.2 The Stage 2D simulator

Stage is a public domain, scalable multiple robot simulator; it simulates a population of mobile robots moving in and sensing a two-dimensional bitmapped environment, controlled through Player. Stage provides virtual Player robots which interact with simulated rather than physical devices. Various sensor mod-
els are provided, including sonar, scanning laser rangefinder, pan-tilt-zoom camera with color blob detection and odometry. Whether using the simulator or real robots, the client control program is always talking to Player, so little or no changes are required to move from simulation to hardware (and back). This is accomplished by using a client-server architecture where the client and the server communicates in a standard manner through TCP/IP connections. Several controllers designed in Stage have been demonstrated to work on real robots. Stage was developed at the USC Robotics Lab [GVS+01].

3.3 The Gazebo 3D simulator
Gazebo is a high-fidelity robot simulator, including physics interaction with the environment. As such, it is more computationally expensive, and simulations are usually restricted to a few robots.

Since the project deals with a swarm of robots, Gazebo will not be used primarily for simulations. However, it could be useful for the development and testing of human robot interfaces, since it provides a more immersive experience to the user.

4 Player / Stage installation and configuration

4.1 System requirements
Player/Stage runs on any modern Linux distribution, as well as on Mac OSX. A native Windows version is planned for release 2.2 (current release is 2.0.4). As for the performance of the simulator, the developers report a 600 MHz Pentium III Linux PC running 200 sonar-guided robots, or 15 laser-guided robots in real time, at spatial and temporal resolutions of 0.02m and 100ms, respectively.

4.2 Install procedure on Linux
The standard installation procedure consists of downloading the source code, configuring, and compiling the modules. It can be quite time consuming due to the library dependences. Thus, we strongly recommend install the precompiled packages instead. This is very straightforward in many Linux distributions like Debian or Ubuntu. In the following, the detailed steps for the latter are explained.

4.3 The Ubuntu Distribution
To install the standard Player/Stage under an Ubuntu system please follow this instructions. First of all you should install Ubuntu itself. Go to http://www.ubuntu.com and get the latest desktop edition iso version available. At the moment of writing this text it is Ubuntu 7.04 (Feisty). Choose a mirror near you get the image. This image is a live distribution and an installation system. You should reboot your system from the CD burned from this iso and once booted you should double-click in the install icon. It will start the system installation and after a few questions and waiting for a while you will get a fresh Ubuntu system.
The second step is to install Player/Stage. The most straightforward way is to use the precompiled packages: download the Player and the Stage “.deb” files from the sourceforge Player/Stage website\(^1\). At the time of writing, they are Player 2.0.4 and Stage 2.0.3. Next, open a console shell (Applications, then Accessories menu), locate the folder where the downloaded files are (usually Desktop) and run the following commands:

```bash
sudo apt-get update
sudo dpkg -i *.deb
sudo apt-get install -f
```

You will be asked for the password of the user you created during the installation process in order to get administration rights. The first command will update the packages repository of the system. The second one will try to install the Player/Stage downloaded files and will fail to finish the process due to dependencies. And finally the third command will fix those dependencies problems. To test your installation issue the following command:

`player /usr/share/stage/worlds/everything.cfg`

If everything works well you should see a screen with some Pioneer robots inside a building (figure 1). You may take control of one of those robots, by running in a new shell terminal:

`playerv`

It will show you a simple test control tool. Subscribe, Command and Enable the position2d:0(stage) device under the Devices menu, and Subscribe the laser:0(stage) device as well. At the end you will be able to command the red Pioneer. Do it by simply click and drag the central red cross of the playerv window to the right (figure 2).

\(^1\)http://playerstage.sourceforge.net

Figure 1: Robots inside a building
4.4 Installing on Macintosh and Windows

Player can be natively installed on Mac OS X (see instructions on the web site), and a native Windows version is expected for release 2.2. Another alternative is to use a virtual machine approach. Under Macintosh, we have tested Parallels\(^2\), and under Windows, VMWare player\(^3\) is a free application, with already created Ubuntu installations\(^4\).

4.5 Advanced installation

The standard Player/Stage installation comes with a bunch of drivers for lots of devices. Although you can use this installation, you could be interested in a more advanced one, choosing some devices for your system that are being just developed. It can be accomplished by checking out the latest CVS files from the sourceforge CVS repository of the project and following the steps suggested in the last part of the FAQ\(^5\).

5 Setting up a scenario

5.1 The world

A world consists of a map of the environment and a set of object models. In figure 3 we can see a reconstructed map of a real environment where five firefighters died in an incident occurred in Worcester (Massachusetts, USA) in 1999 [Ste02]. Firefighters died because they got disoriented while exploring the building and in consequence they ran out of oxygen.

The map is based on a separate bitmap file (white pixels are free space, black pixels are obstacles). The real dimensions in meters need to be defined, as well as the resolution of the underlying bitmap model. Larger values speed up raytracing at the expense of fidelity in collision detection and sensing. Be

---

\(^3\)http://www.vmware.com/products/player/
\(^4\)http://www.thoughtpolice.co.uk/vmware
\(^5\)http://playerstage.sourceforge.net/wiki/Basic_FAQ
sure to create big enough walls and specially corners as the raytracing model will fail with 1 pixel lines.

5.2 The models

The basic model simulates an object with basic properties; position, size, velocity, color, visibility to various sensors, etc. The basic model also has a body made up of a list of lines. Internally, the basic model is used as a base class for all other model types. The basic model can be used to simulate environmental objects.

Other models:

- Blobfinder: The blobfinder model simulates a color-blob-finding vision device, like a CMUCAM2, or the ACTS image processing software. It can track areas of color in a simulated 2D image, giving the location and size of the color 'blobs'. Multiple colors can be tracked at once; they are separated into channels, so that e.g. all red objects are tracked as channel one, blue objects in channel two, etc. The color associated with each channel is configurable.

- Fiducial: The fiducial model simulates a fiducial-detecting device.

- Laser: The laser model simulates a scanning laser rangefinder.

- Position: The position model simulates a mobile robot base. It can drive in one of two modes; either differential, i.e. able to control its speed and turn rate by driving left and right wheels like a Pioneer robot, or omnidirectional, i.e. able to control each of its three axes independently.

- Ranger: The ranger model simulates an array of sonar or infra-red (IR) range sensors.
5.3 The configuration file

The configuration file passed to the Player when launching it defines all the devices if running in a real robot, and all the robots, devices, and the world simulation file, when running Stage simulator. The structure is very simple, made up with sections named “driver” enclosing in brackets the parameters of the desired driver.

driver
(
    name "stage"
    provides ["map:0"]
    model "floor"
)

Each driver provides some services, some of them relaying on a required set of services already provided by other drivers. In the case above the “stage” driver provides a map of the floor (the model “floor” will be described in the Stage world definition). To make the Player start the Stage and so enter in a simulation environment, a driver like this is needed:

driver
(
    name "stage"
    provides ["simulation:0"]
    plugin "libstageplugin"

    # load the named file into the simulator
    worldfile "floor.world"
)

It tells Player to load the Stage plugin, providing the simulation environment, and loading the settings of this simulated world from the file “floor.world” that we will explain later.

5.4 The world definition file

The world definition file is only needed for simulations. It describes models (e.g.: robots, maps, etc), it determinates the size and scale factor of the Stage window, and some other behaviors of the Stage subsystem. What is more interesting and confusing is to define the relation between the map image file and the simulation environment in terms of width and height. If you fail setting up this relation you will come up with a world to small for a very big robot, or in the other side, a robot too small to walk through a really huge world. The size of the world is set using the following line in the world file (be careful as the size is interpreted as meters)

size [58 65]

Then you should select the correct resolution. It is related to the real resolution of the image and the size scale of the map from which you created the bitmap file. It is the resolution of the underlying reytrace model in meters.
resolution 0.1

Then you should configure the initial size and scale factor of the Stage window. It’s mainly for convenience proposes.

window
{
  size [ 1000.000 800.000 ]
  center [-0.010 -0.040]
  scale 0.04
}

5.5 The map file

The map of the world is a bitmap file, usually in png format, where white pixels are free space while black ones are obstacles. To create your own map you will need to create such a file, paying special attention to the image resolution, as it will give you the relation between the map scale and the pixels. For instance, if you have a 1:100cm map in any CAD application, you will need to pay attention when rendering into a bitmap at, let’s say, 100dpi, as you will have 100 pixels per inch, so 100 pixels per each 2,56cm approximately, and so 256cm of reality enclosed in 100 pixels. This relation will be needed, as shown above, in the setup of the configuration file for this world.

5.6 Running a navigation test

Once you have your setup, you will need a Client program to command the robot or robots. This program will have to connect to the desired robot, get the information needed from the sensors and/or the environment (the Stage simulator could provide you this information, if you have to check the correctness of some algorithm), perform the calculations, and send the signals to the actuators. An example of such a client can be found in the laserobstacleavoid.cc file included in the Player distribution. In this example the laser range sensor is used to let the robot walk through the building avoiding obstacles. Is not a very intelligent system the one you get, but it’s a simple way to see the potential of this Player/Stage execution and simulation environment.

In addition, you are able to use almost every actual programming language, as Player has special bindings for the most important languages nowadays.

6 Conclusion and future work

In this article we have validated the use of a simulation and development environment. As a result, a simulation set-up has been presented. The Player / Stage platform has been chosen among the available robot development environments and simulators described in recent surveys. Detailed instructions of the installation and configuration of the software have been presented.

In order to develop a full-functioning demonstrator, the steps for completely defining a user scenario have been provided, based on map data of reported incidents. Example navigation algorithms are presented, which could serve as starting points for more complex swarm algorithms to be developed in future projects.
References


