A Survey and Comparison of Commercial and Open-Source Robotic Simulator Software

Aaron Staranowicz, Gian Luca Mariottini
ASTRA Robotics Lab, Dept. of CSE
University of Texas at Arlington
Arlington,Texas,76019,USA
aaron.staranowicz@mavs.uta.edu
gianluca@uta.edu

ABSTRACT

Simulators play an important role in robotics research as tools for testing the efficiency, safety, and robustness of new algorithms. This is of particular importance in scenarios that require robots to closely interact with humans, e.g., in medical robotics, and in assistive environments. Despite the increasing number of commercial and open-source robotic simulation tools, to the best of our knowledge, no comprehensive up-to-date survey paper has reviewed and compared their features. This survey paper presents a comprehensive and detailed overview and a comparison between the most recent and popular commercial and open-source robotic software for simulation and interfacing with real robots. A case-study is presented, showing the versatility in porting the control code from a simulation to a real robot. Finally, a detailed step-by-step documentation on software installation and usage has been made available publicly on the Internet, together with downloadable code examples.

Categories and Subject Descriptors

A.1 [**General Literature**]: Introductory and Survey- comparison; D.2.6 [**Software Engineering**]: Programming Environments- graphical environments; I.6.3 [**Simulation and Modeling**]: Applications-robotics; I.6.7 [**Simulation and Modeling**]: Simulation Support Systems- environments,portability; I.6.8 [**Simulation and Modeling**]: Types of Simulation- 2-D simulation, 3-D simulation

General Terms

Documentation, Survey

Keywords

Survey, Robotic Simulators, Virtual Environments for Assistive Applications, Open-Source, Player, Stage, Gazebo, ROS, Portability

1. INTRODUCTION

In the recent years we witnessed an emerging need for the development of accurate, robust and easy-to-use software tools for the simulation of robotic models, sensors, and control in virtual environments. This need is motivated by the proliferation of modern robotic applications in which robots are required to autonomously operate in close interaction with humans. This is the case, e.g., in medical robotics [28], in human-robot interaction [21], or in robotics for assistive environments [17]. In these cases, due to the variability of the surroundings, it is imperative to test the safety, efficiency, and robustness of any new algorithm by means of realistic and reliable simulations. Testing the robustness of the performance of a robotic design by modifying only specific environmental conditions is one of the benefits of simulation software. Moreover, code portability from a simulated to a real platform is another feature required in most modern robotic-software tools.

Due to these needs, a growing number of commercial and open-source software-simulation tools have been designed. The robotics community has recognized the need for open-source software for robotic simulation and interfacing [6, 1], and has started to devote increasing interest to these tools and their applicability, as witnessed by the 2009 workshop at the International Conference of Robotics and Automation (ICRA) [6]. Despite the proliferation of robotic simulation software, to the best of our knowledge, there is no up-to-date paper that comprehensively surveys the most widely adopted and recent commercial and open-source software for robotic simulation and interfacing. As a consequence, the novice user (student, researcher, amateur, etc.) will struggle in deciding which software is best suited for his/her purpose. In addition, for some of the existing robotic software tools, there is a need to provide the users with easy-to-use examples, detailed installation and usage instructions.

This survey paper presents a detailed overview and comparison of the most recent and popular commercial and open-source robotic simulation software. We present a case-study to illustrate the portability of an open-source simulator to a real platform. Finally, a step-by-step documentation on software installation and usage has been made available publicly on the Internet, together with downloadable code examples.

2. RELATED WORK

One of the first simulation overview was presented in [18]. Here the authors described the 3-D simulation software Gazebo [8] in detail and presented both case studies to illustrate the use of Gazebo, as well as, a small overview of other software. However, no comparison between their characteristics was provided. In addition, no code example on Gazebo execution, 3-D modeling or interaction was presented.

In his recent work [29], Zlajpah presented a general overview of software simulation tools for robotics. Matlab and other simulation toolboxes are described and simple code examples are provided. Other existing open-source and commercial software tools are introduced, but no comparison between their performance is reported and no code example is provided for the novice user.

In [20], the authors presented a description of the open-source Player, Stage, Gazebo (PSG) [8] software package, together with a description of four other robotic software libraries. PSG is used to verify in simulation high-level control algorithms, which are then validated on a real robotic platform. However, no comparison between PSG and the other four software libraries was presented.

In [19], the authors presented a detailed description and comparative study of nine open-source robotic development environments. The comparison between each environment follows a set of evaluation criteria based on software-development process. The authors described performances and usability of each open-source software tool, but do not include commercial software. Moreover, no documentation or code examples were provided to support the beginners.

A recent review of robotic software simulation and programming tools has been presented by Somby in [27]. The author compared the characteristics of eight well-known software packages. However, the software architectures are not presented, and code and usage instructions are not provided. Moreover, [27] is an online publication and has not been peer-reviewed for publication in major robotics conferences or journals.

As an original contribution with respect to the existing literature, we present a general overview and comparison of eleven most-recent robotic simulation software (see Sect. 3), as well as a detailed description of the architecture for two of the most widely used open-source software (see Sect. 4). In addition, Sect. 5 presents a case-study for robotic simulation and interfacing on a real robotic platform.

This paper wants to be of help to beginners who are looking for a simulators that is more suited for their application. As described in the introduction, we realized a step-by-step tutorial documentation and made it publicly available on the Internet to provide installation instructions and code-base examples for the beginners. (c.f., Sect. 4.3)

3. OVERVIEW OF ROBOT-SIMULATION SOFTWARE

We here provide an overview of the most-recent and widely used robot-programming and simulation software systems. We will divide them in two categories: *commercial* and *open-source*. A summary of their characteristics is provided in two tables in Sect. 3.3.

3.1 Commercial Software Packages

Webots [24] is a 3D simulation environment used to model, simulate, and program mobile robots. The simulated models have many different and customizable attributes such as texture, mass, friction, and shape. These attributes are provided by the Open Dynamics Engine (ODE) Library [13] which is used for simulating rigid body dynamics. The simulation can model several types of robotic platforms such as mobile robots (e.g., Epuck, Pioneer, Lego Mindstorm), quadrupeds robots (e.g., Sony's AIBO), and humanoid robots (e.g., Fujitsu's HOAP-2). Webots can simulate a wide number of sensors commonly used in robotics, such as proximity sensors, light sensors, touch sensors, GPS, accelerometers, lasers, and cameras, to name a few. Webots supports a graphical interface to allow users to interact with the environment, robots, or other objects (e.g., furniture). The built-in motion editor is used to create and store a sequence of motions for articulated robots which can be played by the robot controller. Webots can be used to create AVI or MPEG movies of simulations for the web or presentations. The user can simulate a single robot or multiple robotic systems with communication facilities. The programming language used by Webots for creating the control program is C, C++, Java, Python, and Matlab. Webots can interface with third party software through TCP/IP such as LabView. Another peculiarity of Webots is the possibility of transferring the control code to commercially-available real robots via Bluetooth. Users are able to use in the control code high-level algorithms such as SLAM and path planning. Webots supports professional or an educational version for Linux, MAC OSX, and Windows.

Easy-Rob [25] is a planning and simulation software for robotic platforms in manufacturing plants that operate within work cells. *Easy-Rob* allows the user to program and visualize in 3D several processes, such as handling, assembly, coating and sealing. Another functionality of Easy-Rob is the ability to create AVI movies of the simulations. Users can use the API provided by Easy-Rob or external robotic libraries; built-in tools allow the user to import or export AutoCAD designs. Easy-Rob supports multiple- or single-robot environment for Microsoft Windows XP, Vista, and Windows 7.

MATLAB with Simulink [11] is a multi-domain simulation and model -based design for dynamic and embedded systems. The community created Simulink toolboxes in which blocks defined for robotic platforms and sensors can be used. MATLAB and Simulink can be jointly used, together with the free *Robotics Toolbox* (Rob.Tbx.) by P. Corke [2] in order to design simulations with robot manipulators. The Robotics Toolbox provides functions to generate trajectories and analyze results from the simulations and real robots. Simulink and MATLAB are available for Linux, Windows, and Mac OSX systems.

3.2 Open-source Software Packages

Player/Stage (PS) [8] is a free-software project, released under the GNU General Public License, that provides a set of advanced tools for single- and multi-robot interface and control. Player is a device-independent server that provides network-transparent robot control. Player supports connections of multiple clients (e.g., control programs) to single or multiple devices (e.g., robotic platforms, lasers, cameras) via a network connection using TCP/IP sockets. This allows clients to be written in any programming language that provides TCP socket support. Client libraries and APIs have been developed for users to write control code in C, C++, Tcl, Python, Java, Common Lisp, and Matlab. Several community-supported libraries were written to allow Player to support different programming languages such as Ada, Java, and GNU Octave. A broad range of robotic platforms and sensors is supported in Player. Player is available for Windows, Linux, and Mac OSX.

Stage is capable of 2-D simulation of both robotic platforms and of a variety of sensors. Stage provides computationally cheap robotic models and allows for many robots (potentially hundreds of thousands) to be simulated at the same time; it also provides collision avoidance and map generation. Stage attempts to run in real-time by default [4], but it will run slower than real-time if robotic models take longer to update. Stage is available for Linux and Mac OSX. Further details on Player/Stage will be provided in Sect. 4.1.

Gazebo [8] is a 3-D simulator developed to be used with Player. All the functions used in Player/Stage can be used in Gazebo without any modification. Gazebo relies on the Object-Oriented Graphics Rendering Engine (OGRE), [14] and ODE, [13] to render 3D objects (i.e., robots and environments). These libraries allow Gazebo to accurately reproduce the dynamic 3-D environments a robot may encounter [18]; all simulated objects have mass, friction, and numerous other attributes that allow them to behave realistically when pushed, pulled, knocked over, or carried. The simulated robots can be assembled using multiple shapes with different joints between each shape; this allows the user to model a wide range of robotic platforms. The environment is as complex as the simulated robots and it can be assembled using simulated light and static objects (e.g., buildings, furniture, and landscapes). Gazebo is available for Mac OSX and Linux. Further details will be provided in Sect. 4.1.

Robot Operating System (ROS) [15] is an open-source operating system which provides hardware abstraction, low-level device control, message-passing between processes, and package management. One of the most interesting characteristics of ROS is the presence of a broad and growing community of researchers contributing to its expansion. This is possible through the presence of many code repositories available on the ROS website. ROS supports four different languages: C++, Python, Octave, and LISP [22]. ROS supports a large number of tools that perform a wide range of tasks (e.g., navigation, connection handling, message passing between processes). When implementing control code in ROS, users use nodes, messages, topics and services to perform tasks. ROS is available for Linux, Mac OSX, and on Windows (with limited functionality). Further details will be provided in Sect. 4.2.

Simbad [26] is a Java 3-D simulator for single or multiple robots designed to study AI and machine learning in robotics. Simbad supports vision sensors, range sensors, and contact sensors. Simbad provides a Neural Network library and an Evolutionary Algorithms library. The supported programming languages are Java and Python. Simbad is available for Mac OSX, Windows and some Linux distributions.

Carnegie Mellon Robot Navigation Toolkit (CARMEN) [7] is a collection of software for mobile robot control. CARMEN is designed to provide the user with an easy access to robotic algorithms such as base and sensor control, logging, obstacle avoidance, localization, path planning, and mapping. CARMEN features robot control software, 2-D robot simulation, and supports the interface of several real mobile-robot platforms and sensors (e.g., range sensors, proximity sensors, and GPS). CARMEN supports C and Java. Linux is the only currently supported operating system.

Unified System for Automation and Robot Simulation (USARSim) [3] is a 3D simulator based on the Unreal Tournament game engine. USARSim was created to simulate urban search and rescue robots and environments intended for the study of human-robot interaction and multi-robot coordination. The control code can be written using the GameBot interface which is the Unreal engine proprietary communication protocol, the Mobility Open Architecture Simulation and Tools system (MOAST), Player interface, or the USARSim Matlab Toolbox [9]. The controller (e.g., Player, MOAST) connects to the Unreal server, which sends commands to USARSim to create a robot model. The controller listens for sensor data and sends commands to control the robot. The most important part of USARSim is the environment the robots move in; there are several default maps that are in the Urban Search and Rescue domain but more can be created by users. A variety of sensors are supported such as touch sensors, sound sensors, cameras, and lasers. USARSim supports Windows and Linux.

Microsoft Robotics Developer Studio (MRDS) [12] is a Windows-based 3-D simulator to create robotic applications across a variety of robotic platforms and sensors. MRDS is not technically an open source simulator, but it is available to the public at no cost and includes the functionalities of all the editions and the toolkits. MRDS uses a visual programming language (VPL) to create the robotic applications and control by connecting together a collection of blocks. The blocks can be reused over again based on the inputs and outputs, and can be manually edited. MRDS can connect to either the Visual Simulation Environment or a real robotic platform. The Visual Simulation Environment is the 3D simulation tool that uses NVIDIA PhysX and Microsoft XNA Framework to provide both the physic behavior and the real-time 3D graphics rendering. A variety of sensors is supported and accessed by services that MRDS provides at runtime.

MissionLab [10] is a Linux-based 3-D simulator for multi-agent robotics mission specification and control software. MissionLab uses high-level military-style plans and executes them on teams of simulated or real robotic platforms. Several tools are used to plan the missions and write the configurations for each robotic platform through a graphical interface.

We are aware that many other robotic software packages exist. However, as explained at the beginning of this section, we are interested herein, only in those that are more widely used by researchers and students.

3.3 Summary of Robotic Software

In Table 1 and Table 2, we present a summary of the main features of the commercial and open-source simulators (see Sect. 3.1 and Sect. 3.2). The software systems are compared according to the following parameters. (i) *Operating system (OS)*; it describes which OS is supported by the robotic software. (ii) *Simulator type*; it describes if the robotic software provides either a 2-D or 3-D simulation environment. (iii) *Programming language*; it describes the language supported by the robotic software. (iv) *Documentation*; it describes the level of documentation available with the robotic software and can either be "high-level" or "low-level". "High-level" means the documentation only provides descriptions of the functions in the robotic software libraries; "low-level" means the documentation provides the code for the functions in the robotic software libraries. (v) *Tutorial*; it describes if examples and a step-by-step guide are provided. "Yes" means that a well defined guide with examples is available; "limited" means a guide exists, but not enough details and examples are provided. Finally, "No" means

	Webots	**Easy-Rob**	**Rob.Tbx.**
OS	Mac, Linux, Win.	Win.	Mac, Linux, Win.
Simulator Type	3-D	3-D	2-D, 3-D
Programming Language	C, C++, Java, Matlab, Python	C++	C-like
Documentation	High-Level	High-Level	High-level
Tutorial	Yes	Yes	Limited
Portability	Yes	No	No
Sensors	odometry, range, camera, GPS	odometry	odometry, range
Debugging/ Logging	Yes	Yes	Yes
Graphical User Interface	Yes	Yes	Yes

Table 1: Characteristics of Commercial Robotic Simulation Softwares (c.f., Sect. 3.3)

	Player/Stage	Gazebo	ROS	Simbad	CARMEN	USARSim	MRDS	MissionLab
OS	Linux, Mac, Win.	Linux	Linux, Mac, Win.	Linux, Mac, Win.	Linux	Linux Win.	Win.	Linux
Simulator Type	2-D	3-D	2-D, 3-D	3-D	2-D	3-D	3-D	3-D
Programming Language	Player(any) Stage(C, C++, Python, Java)	C, C++, Python, Java	C++, Python, Octave, LISP, Java, Lua	Java	C, Java	C, C++, Java	VPL, C#, Visual Basic, JScript, Iron-Python	VPL
Documentation	Low-Level	Low-Level	High-level	High-Level	Low-Level	High-Level	High-Level	High-Level
Tutorial	Yes	Yes	Yes	Limited	Limited	Limited	Yes	Limited
Portability	Yes	Yes	Yes	Limited	Yes	Yes (using Player)	Yes	Yes
Sensors	odometry, range	odometry, range, camera	odometry, camera, range	vision, range, contact	odometry, range, GPS	odometry, range, camera, touch	odometry, range, camera	odometry, range
Debugging/ Logging	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
Graphical User Interface	No	No	No	No	No	Yes	Yes	Yes

Table 2: Characteristics of Open-Source Robotic Simulation Softwares (c.f., Sect. 3.3)

there is not a useful tutorial and/or examples. (vi) *Portability*; "Yes" means that the code written for a simulation is portable to a real robotic platform. (vii) *Sensors*; it describes the supported sensors. Only the most requested sensors are reported in the table due to space limitations. (viii) *Debugging/Logging* describes if debugging, fault tolerances, play-back, and logging features are provided by the robotic software. (ix) *Graphical User Interface*; it describes if it is possible to modify objects and the environment during run-time and/or program functions in an development environment. The graphical user interface does not include windows that open to display the simulation.

4. ARCHITECTURE

In this section, we provide a detailed description of both the system architecture and the main features of the two most popular *open-source* software for robotic simulation: Player /Stage /Gazebo and ROS. These two simulators, in fact, exhibit unique features, such as accurate 3-D environment modeling, as well as the possibility to interface with real robots and sensors. We realized a detailed step-by-step documentation on the installation and usage of Player/Stage/Gazebo and ROS (see Sect 4.3) which will be of help to students and researchers who are approaching these softwares for the first time.

4.1 Player/Stage/Gazebo

As mentioned in Sect. 3 and as shown in Figure 1, Player is a device-independent server that provides network-transparent robot control over TCP/IP sockets. Clients (e.g., control programs) can be written in any programming language that supports TCP/IP and can connect to any device (e.g., lasers, cameras) through Player.

Player creates the connections between clients and devices based on a *configuration file* that contains information on which drivers need to be created and how to bind them to the devices. Even if clients receive data at a default global rate of 10Hz [5], they can configure Player to increase or decrease the data-rate to match one of the devices.

Multiple clients can subscribe to the same device and can write simultaneous commands but there is no queue and old commands are disregarded. However, clients must have the proper access-rights to send or receive data from the device, since Player treats devices as files [4].

Herein Stage will be described as a plug-in for Player, even if it could be used in other two different ways; as a stand-alone program (in which a user provides the library), or as a part of a user's own program. Stage supports blob detection algorithms along with several types of sensor models (such as lasers, sonars, and position). Stage also allows for simulating non-existent sensors, such as lasers that can pass through walls.

The environment parameters, the robotic platforms, and the sensors simulated in Stage are defined in a *world file*. The environment visualized in the simulation uses PNG image files as input that show a black/white top view of the obstacles inside the environment.

The simulation using Gazebo is defined by the world file which is an XML text file. As mentioned in Sect. 3.2, Gazebo renders 3-D models due to the OGRE and ODE libraries. ODE is designed to simulate rigid body dynamics and includes features such as joints, collision detection, mass, and rotational functions. OGRE renders the robotic models, environment, and objects in the simulation. The robotic models are created from rigid bodies and joints. The rigid bodies are generated by composing primitive shapes such as cubes, spheres, and cylinders and each one has an assigned mass, friction, color and texture. The joints connect the bodies together and form

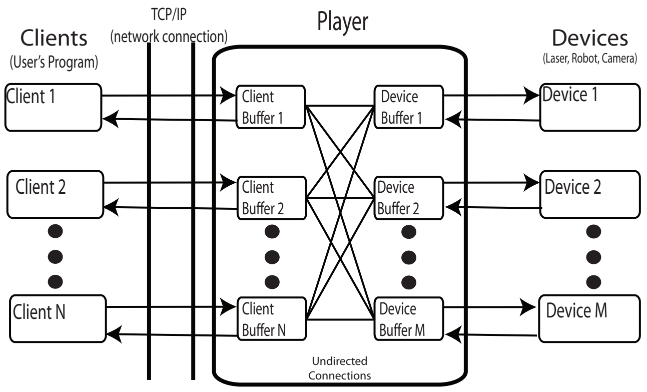

Figure 1: The diagram shows the connectivity between the clients and the devices in Player. Player allows multiple clients to connect to multiple devices.

relationships between the bodies. There are several types of joints such as universal joints, ball and socket joints, and hinge joints. Figure 2 shows the relationship between ODE, OGRE 3D, shapes, joints, and the environment.

Player, Stage, and Gazebo do support high-level algorithms such as blob tracking, localization methods and mapping [19]. Due to the abstraction used in Player, control code can be used either on a simulated, or on a real robot. An additional functionality of Player is the "pass-through" driver, that allows independent Player servers to act like clients and connect to other Player servers [4]. The computational complexity of Gazebo in simulating rigid-body dynamics and 3-D environments can severely tax even a high performance computer; this limits Gazebo to simulate only a few robotic platforms [18].

4.2 ROS

Robot Operating System (ROS) provides operating system-like tools (e.g., message-passing between processes) and package tools

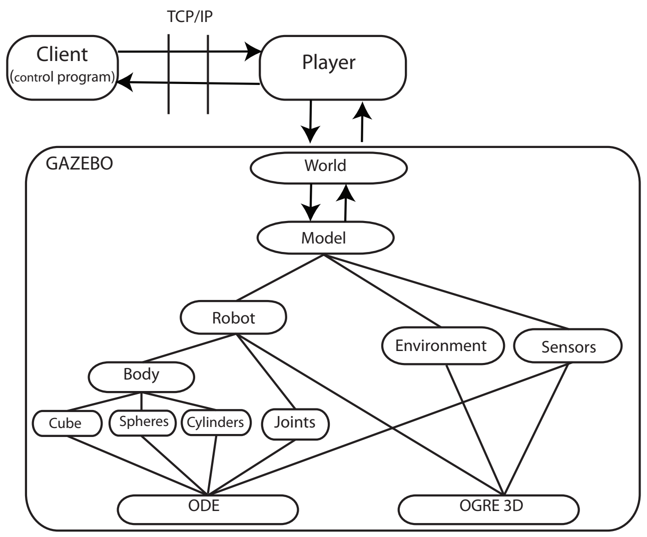

Figure 2: The diagram shows the connectivity between the Client, Player, and Gazebo

(e.g., create and find ROS packages). ROS is based on nodes, messages, topics, and services. Nodes are processes or software modules in the control code (e.g., a camera node could process all of the visual data). Nodes can communicate with other nodes by passing simple messages (or data structures). Nodes publish and subscribe to a single (or multiple) topics. ROS supports TCP/IP and UDP for message passing; a special type of message, called service, consists of a pair of messages, one for request and the other for reply. The ROS Master keeps track of all services and topics; it also provides node registration and a parameter server which allows nodes to store and retrieve parameters. The ROS Master server and tools are configured with XML files.

ROS has several client libraries (e.g., control programs) available such as *rosccp* (C++ library), *rospy* (Python library), *rosoct* (Octave library), *roslisp* (LISP library), *rosjava* (Java library), and *roslua* (Lua library). Each library uses a set of ROS tools to facilitate the development of new clients in ROS. The client libraries are organized into packages. Packages (organized software modules) contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party software, and/or other useful modules. Packages allow for code reuse but stacks, a collection of packages, allow for code sharing.

The ROS tools allow a easy usage of packages and stacks and are divided in to three categories: file system, command-line, and logging. The common file system tools allow users to perform tasks in ROS's file system such as *rospack*, *roscd*, *rosmake*, and *roscreate*. The common command-line tools such as *roscore*, *rosrun*, *roslaunch*, and *rosservice* allow users to execute, initialize, or provide details about nodes and services. The logging tools allows users to debug ROS packages and stacks such as *rosbag* which records and playbacks of ROS topics. There are several visual command tools such as *rxbag* and *rxbag plugins* that are used for visualizing the contents that are stored by the rosbag command. *Rviz* is a 3D visualization environment for robots using ROS and allows *rxbag plugins*. *Rviz* has a graphical user interface to allow users to configure and modify objects and the environment.

The advantage of ROS is code reuse and sharing. Code sharing allows everyone to have a common basis and helps with replicating and testing the source code. By using the ROS tools, implementation is efficient and simplifies some of the complex tasks needed to execute the code but this creates a dependency to ROS and reduces the mobility of the code. ROS can connect to real robots and use simulated robots in ROS's Gazebo, Stage, and Rviz. However, control code written for Player's Stage and Gazebo will have to be modified to work in Gazebo/Stage under ROS.

4.3 Documentation and Source Code

In order to allow the novice user to rapidly work with the above software, we realized a detailed step-by-step documentation containing installation instructions and downloadable code examples.

The current version of the documentation presents two robotic software simulators and, for each one of them, proposes a description, an installation guide, and simulation examples for interaction with a widely used robotic platform (Create, by iRobot). The website can be found here:
http://ranger.uta.edu/~gianluca/astra/projects/robotsimulation/index.html.

5. A CASE STUDY

In order to illustrate the versatility of Player we consider a trajectory-tracking task for a mobile robot. In particular, we show that minor modifications to Player's configuration files are needed to port the control code from the Gazebo simulation to the real robotic platform.

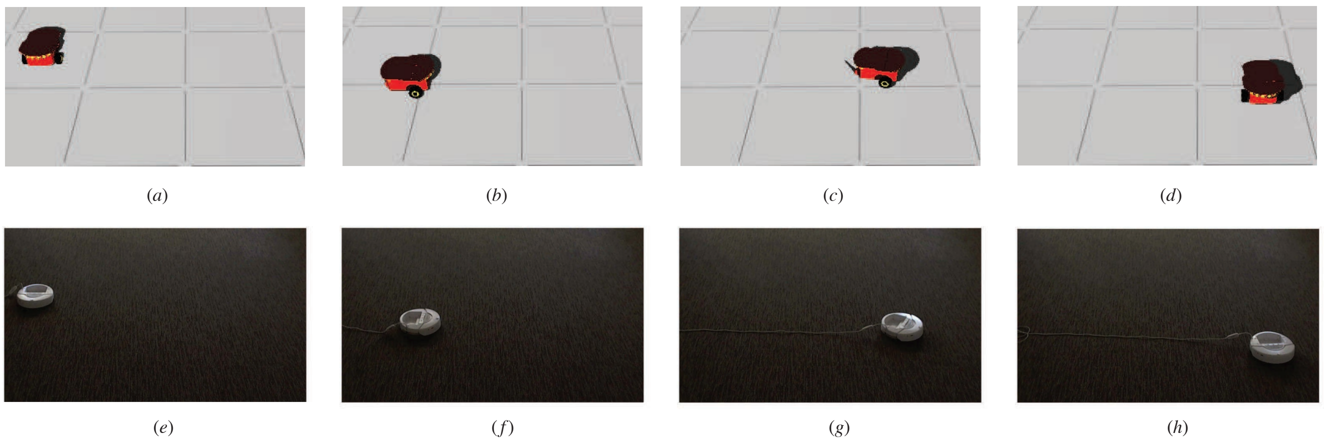

Figure 4: *Top row*: Pioneer in Gazebo. *Bottom row*: iRobot Create. The Create and Pioneer use the same trajectory planning and tracking algorithm from the initial pose $\mathbf{x}_i = [0m \;\; 0m \;\; 0rad]^T$ to the final pose $\mathbf{x}_f = [1m \;\; 2m \;\; 0rad]^T$. Very few changes to the *configuration file* are necessary to adapt the entire control algorithm from the simulated to the real-robot scenario.

5.1 Hardware and Software Setup

We used an iRobot Create [16] and a Linux-based netbook. We used the serial port of the Create to connect to the Linux-based netbook via USB-to-Serial cable. We used Player v.3.0.2 and Gazebo v.0.9.0 (with ODE v.0.11.1 and OGRE 3D v.1.6.4). Gazebo was setup to use the default Pioneer model, a flat, empty world and two light sources which provide a directional light and a point light. The point light radiates light to the entire world and the directional light radiates light to the Pioneer model.

Player instantiates drivers based on two configuration files, one for the iRobot Create and the other for Gazebo (Table 3). The Create configuration file has one driver which contains the *name* of the driver, the *sensor* provided by the robot, the *communication port* (i.e., the name of the USB port connected to the robot), and if the driver should start when Player starts or waits for the client to connect. Gazebo requires two drivers to be instantiated by Player. The first driver defines the *name* of the driver, the filename of the *plugin*; in our specific case is `libgazeboplugin`, i.e., Gazebo simulation library and the communication port which is the default port (i.e., localhost). The second driver provides the name of the driver, which sensors the robot provides, and the identification of each sensor within Gazebo.

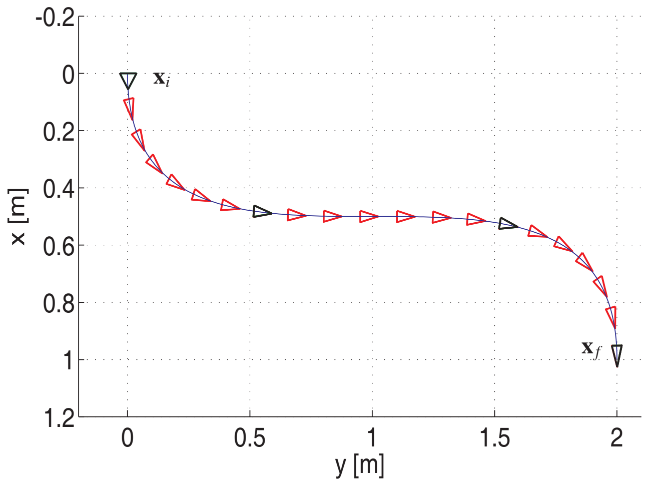

Figure 3: The planned trajectory from $\mathbf{x}_i$ to $\mathbf{x}_f$.

Player config. for Create:
```
driver
(
name "create"
provides ["position2d:0"]
port "/dev/ttyUSB0"
alwayson 1
)
```
Player config. for Gazebo:
```
driver
(
name "gazebo"
provides ["simulation:0"]
plugin "libgazeboplugin"
server_id "default"
)
driver
(
name "gazebo"
provides ["position2d:0"]
gz_id "pioneer2dx_model1::position_iface_0"
)
```

Table 3: Player configuration for the Create and Gazebo

5.2 Trajectory Planning

This section illustrates the of trajectory planning and tracking via Cartesian polynomials [23]. The goal of the trajectory planning is to provide a reference smooth curve that will lead the robot from an initial to a final pose, $\mathbf{x}_i = [x_i \;\; y_i \;\; \theta_i]^T$ and $\mathbf{x}_f = [x_f \;\; y_f \;\; \theta_f]^T$, respectively, (see Figure 3). This parameterized curve (in s) satisfies the nonholonomic constraints of our unicycle-like robotic platform. The analytical expression for the curve is given by the following:

$$x(s) = s^3 x_f - (s-1)^3 x_i + \alpha_x s^2(s-1) + \beta_x s(s-1)^2,$$
$$y(s) = s^3 y_f - (s-1)^3 y_i + \alpha_y s^2(s-1) + \beta_y s(s-1)^2,$$
$$\theta(s) = Atan2(\dot{y}(s), \dot{x}(s)) + k\pi, \quad k \in \{0,1\}, s \in [0,1].$$

The parameters $\alpha_x, \alpha_y, \beta_x, \beta_y$ are obtained by imposing the boundary conditions on the unicycle model:

$$\alpha_x = k_f \cos(\theta_f) - 3x_f \quad \alpha_y = k_f \sin(\theta_f) - 3y_f,$$

$$\beta_x = k_i \cos(\theta_i) + 3x_i \quad \beta_y = k_i \sin(\theta_i) + 3x_i.$$

In our experiment we picked $k_i = k_f = 3$.

5.3 Trajectory Tracking

The reference trajectory calculated in the previous section, will be used by the tracking control law to allow the robot to move from its current pose $[x(t), y(t), \theta(t)]$ to the final pose $\mathbf{x}_f$. We implemented a control law based on input/output feedback linearization [23]. The velocity input $\mathbf{u} = [v,\ \omega]^T$ of the unicycle is obtained by:

$$\mathbf{u} = \mathbf{T}^{-1}(\dot{\mathbf{y}}^{des} - \mathbf{K}(\mathbf{y} - \mathbf{y}^{des})),$$

where,

$$\mathbf{T}^{-1} = \begin{bmatrix} \cos\theta(t) & \sin\theta(t) \\ (-\sin\theta(t))/b & (\cos\theta(t))/b \end{bmatrix},$$

$$\mathbf{K} = \begin{bmatrix} k_1 & 0 \\ 0 & k_2 \end{bmatrix}, \quad k_1\ ,\ k_2 > 0,$$

and

$$\mathbf{y} = \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} x(t) + b\cos\theta(t) \\ y(t) + b\sin\theta(t) \end{bmatrix},$$

where $b > 0$ represents a point along the sagittal axis of the unicycle with a distance b from the contact point of the wheel to the ground [23]. The desired trajectory is based on the unicycle model using the following equations:

$$\mathbf{y}^{des} = \begin{bmatrix} y_1^{des}(t) \\ y_2^{des}(t) \end{bmatrix} = \begin{bmatrix} x(s) + b\cos\theta(s) \\ y(s) + b\sin\theta(s) \end{bmatrix}.$$

Figure 4 shows the performance of the trajectory tracking. Figs. 4 (*a*), (*e*) show the initial pose of the robot in both the simulated and real robot. The robot moves from the initial pose along the curve (Figs. 4 (*b*), (*f*) and Figs. 4 (*c*), (*g*)). The robot stops once the final pose is reached (Figs. 4 (*d*), (*h*)).

6. CONCLUSIONS AND FUTURE WORK

In this paper we presented a detailed overview and comparison of eleven commercial and open-source software tools for robotic simulation and real-platform interfacing. Due to space limits, we presented an in-depth description of the architecture of two software systems. We illustrate the versatility of Player by presenting a trajectory planning and tracking code example that works in Gazebo, and needs minor modifications to work on the robotic platform. We created a step-by-step documentation that describes the installation and usage of these robotics tools, and we made it publicly available on the Internet, along with downloadable code-examples. This paper along with the detailed documentation will allow beginners to gain comprehensive knowledge of the available robotic simulation software.

As a future work, we will expand our survey to other software systems, and include additional case studies. Finally, we plan to expand the number of code-examples provided on the project website.

7. ACKNOWLEDGMENTS

We would like to thank Dr. Fabio Morbidi for his invaluable support in the preparation of the final manuscript.

8. REFERENCES

[1] H. Bruyninckx. Robotics software: The future should be open. *IEEE Robotics Automation Magazine*, 15(1):9 –11, March 2008.

[2] P.I. Corke. A computer tool for simulation and analysis: the robotics toolbox for matlab. In *Proceedings of the 1995 National Conference of the Australian Robot Association*, pages 319–330, July 1995.

[3] Unified System for Automation and Robot Simulation [Online]. Available: http://usarsim.sourceforge.net/wiki/index.php.

[4] B.P Gerkey, R.T Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th International Conference on Advanced Robotics*, pages 317–323, 2003.

[5] B.P. Gerkey, R.T. Vaughan, K. Stoy, A. Howard, G.S. Sukhatme, and M.J. Mataric. Most valuable player: A robot device server for distributed control. In *2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1226 –1231, 2001.

[6] H. Hirukawa and A. Knoll. ICRA 2009 Workshop on Open-Source Software in Robotics. [Online]. http://www.openrtp.jp/icra09_workshop/.

[7] Carnegie Mellon Robot Navigation Toolkit [Online]. Available: http://carmen.sourceforge.net/.

[8] Player/Stage/Gazebo [Online]. Available: http://playerstage.sourceforge.net/.

[9] USARSim Matlab Toolbox [Online]. Available: http://robotics.mem.drexel.edu/USAR/index.html.

[10] MissionLab [Online]. Available: http://www.cc.gatech.edu/ai/robot lab/research/MissionLab/.

[11] Matlab [Online]. Available: http://www.mathworks.com/products/simulink/.

[12] Microsoft Robotic Developer Studio [Online]. Available: http://www.microsoft.com/robotics/.

[13] Open Dynamics Engine [Online]. Available: http://www.ode.org/.

[14] Object-Oriented Graphics Rendering Engine [Online]. Available: http://www.ogre3d.org/.

[15] Robot Operating System [Online]. Available: http://www.ros.org/wiki/.

[16] iRobot Create [Online]. Available: www.irobot.com/create/.

[17] Dae-Jin Kim, R. Lovelett, and A. Behal. An empirical study with simulated adl tasks using a vision-guided assistive robot arm. In *IEEE International Conference on Rehabilitation Robotics*, pages 504 –509, June 2009.

[18] N. Koenig and A. Howard. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2149–2154, 2004.

[19] J. Kramer and M. Scheutz. Development environments for autonomous mobile robots: A survey. *Autonomous Robots*, 22:101–132, February 2007.

[20] N. Michael, J. Fink, and V. Kumar. Experimental testbed for large multirobot teams. *IEEE Robotics Automation Magazine*, 15(1):53–61, March 2008.

[21] R.R. Murphy. Human-robot interaction in rescue robotics. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 34(2):138 –153, May 2004.

[22] M. Quigley, K. Conley, B.P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A.Y. Ng. ROS: an open-source robot operating system. In *International Conference on Robotics and Automation Workshop on Open Source Software*, 2009.

[23] B. Siciliano, L. Sciavicco, and L. Villani. *Robotics: Modelling, Planning and Control*. Advanced textbooks in control and signal processing. Springer, 2009.

[24] Webots: 3D simulation [Online]. Available: http://www.cyberbotics.com/overview.

[25] Easy Rob: 3D simulation [Online]. Available: http://www.easy-rob.com/en/easy rob.html.

[26] Simbad: Java 3D simulator [Online]. Available: http://simbad.sourceforge.net/.

[27] M. Somby. Software platforms for service robotics, 2008. [Online]. Available: http://www.linuxfordevices.com/c/a/Linux-For-Devices-Articles/Updated-review-of-robotics-software-platforms/.

[28] R.H. Taylor and D. Stoianovici. Medical robotics in computer-integrated surgery. *IEEE Transactions on Robotics and Automation*, 19(5):765 – 781, Oct. 2003.

[29] L. Zlajpah. Simulation in robotics. *Math. Comput. Simul.*, 79:879–897, Dec 2008.