Introductory Survey to Open-Source Mobile Robot Simulation Software

Patricio Castillo-Pizarro†, Tomás V. Arredondo† and Miguel Torres-Torriti‡

***Abstract*— Mobile robot simulation is a valuable tool for education, research and design purposes. The last decade has seen a considerable increase in the development of new software tools for mobile robot simulation, all of which have reached different levels of maturity. This paper presents a survey of the existing tools and discusses their strengths and drawbacks in terms of simplicity, flexibility, fields of application, among other desirable features. This discussion should be valuable to students, teachers, engineers and researchers alike who are seeking adequate tools for simulating autonomous mobile robots. Introductory examples showing the usage of three of the most mature and widely used open-source simulators (Carmen, Gazebo and Open Dynamics Engine) are also included.**

I. INTRODUCTION

Simulation of robotic systems is an essential tool in teaching, research, and development [1], [2], [3], [5], [4], [9]. Without simulation tools it would be hard, if not impossible, to design and test the feasibility of complex systems like closed kinematic chain robots [6] or humanoid robots [7], explore design options, assist in off-line programming of robot systems [3], evaluate the performance of robotic cells [4], develop virtual environments for operator training or teleoperation, nor test advanced robot control algorithms [2]. The role of simulation is also important as a means to facilitate teaching and learning the fundamental physics and the underlying concepts of robotic systems; which could otherwise be very hard to grasp without actual experimentation and visualization [1], [8]. Robotic systems can be costly, and therefore, simulation provides an excellent alternative of doing experiments and testing systems away from the dangers and unpredictability of the physical world [1].

Although simulation has been important since the beginning of robotics, its development has been tightly related to the developments in computing. The massification of personal computing during the 90s contributed to the development of several publicly available software tools for robotic manipulator modeling and simulation. Some of these tools included: XAnimate [8], the Robotics Toolbox for Matlab [9], and the Robotica package for Mathematica [10]. During the late 90s, the focus of attention shifted towards manipulation, grasping and mobile robots. This trend led to the development of GraspIt! [5], a simulator for robotic grasping, and numerous tools for simulating and interfacing with various mobile robot platforms. The list of programming libraries, simulation tools and packages can be quite long, however, among those open-source tools that have achieved some degree of sustained development it is possible to mention Breve [21], Carnegie Mellon Robot Navigation Toolkit (Carmen) [14], the Open Dynamics Engine (ODE) [19], OpenRave [22], OpenSim [23], the Player-Stage-Gazebo (PSG) project [16], RoboCode [24], Simbad [13] and UsarSim [25]. The are also a number of proprietary tools for mobile robot simulation: Aria [26], Cogmation [27], anyKode Marilou [28], MS Robotics Studio [29], Webots [30].

Researchers, developers and educators faced with this long list of simulation tools may have a hard time to decide which is the right tool for mobile robot simulation, and may not even be aware of all the options available. The decision may not be obvious, some tools may be user friendly and simple, involving only kinematic models and intended for demonstrating or testing autonomous agent control strategies. Other tools may include accurate physical simulation engines, but many depend on different external packages, which makes it more difficult for the user to grasp the details of all the components of the simulation; not to mention that reusability of developed code might be more susceptible to changes in the external libraries causing the software to be hard to maintain and update. Finally, some tools may have high initial costs or even periodic license maintenance fees in order to utilize the software. This is troublesome when engineering robotic systems in developing nations which many time do not have sufficient resources to pay such costs.

Considering these facts, the purpose of this paper is to provide a brief overview of existing open-source mobile robot simulation tools, and assess accuracy, flexibility, user-friendliness, and modularity aspects.

The paper is organized as follows, section II presents a an introductory background on simulator architecture issues. The review of the existing mobile robot simulators is presented in section III, followed by simulation results in IV. Final remarks are presented in section V.

II. SIMULATOR ARCHITECTURE

In order to compare the different elements of simulation software it is fundamental to consider their underlying software architecture. In particular those aspects which allow the software to be modular and run in real-time, possibly distributed over a network of host machines. Roughly, simulators can be classified as standalone applications in which several functions are compiled into one executable linked against the simulator library, or as multi-process systems. In multi-process simulator architectures there is a need for ways to pass information between modules. The main approaches to exchanging data between processes are: file-based memory, RAM-based memory and inter-process communication using message passing mechanisms (e.g. Unix sockets and TCP/IP). The first two approaches can be used to create client-server applications, as well as blackboard systems

† is with the Electronics Department,Universidad Técnica Federico Santa María, Valparaiso, Chile. patricio.castillo@ieee.org, tomas.arredondo@usm.cl,

‡ is with the Dept. of Electrical Engineering,Pontificia Universidad Católica de Chile, Santiago, Chile. mtorrest@ing.puc.cl

based on publish/subscribe mechanisms. The third approach naturally lends itself to applications of the client-server type rather than blackboard architectures. For additional details on these three types of information sharing between process the reader is referred to [15].

Before reviewing some important aspects in each approach it is important to understand the general components and data flow in a simulator. To this end, consider a generic simulation process, which starts with an initialization step of a model and environment description parameters loaded from a file. While stopping conditions are not met, the simulation loops through the model update equations given by

$$\dot{\mathbf{x}}_k = \mathbf{f}(\mathbf{x}_k, \mathbf{u}(\mathbf{x}_k, t_k), t_k) \quad (1)$$

$$\mathbf{x}_{k+1} = \int_{t_k}^{t_k+\Delta t} \dot{\mathbf{x}}_k dt + \mathbf{x}_k \quad (2)$$

$$t_{k+1} = t_k + \Delta t \quad (3)$$

where $\mathbf{x}_k$ is the state vector of the system at time t_k. Equation (1) defines the model in terms of some algebraic description, represented here by a function $\mathbf{f}$, which expresses the rate of change of the state $\dot{\mathbf{x}}_k$ as a function of the current state $\mathbf{x}_k$, a control command $\mathbf{u}(\mathbf{x}_k, t_k)$ that may be implemented as a feedback of the state $\mathbf{x}_k$ possibly time-varying. Equations (2) and (3) update the state value and step time t_k by Δt, and thus are also referred to as stepping functions which advance the solution from time t_k to $t_k + \Delta t$ for a fixed step-size Δt.

The system of equations (1)–(3) can be implemented into a single executable or divided into modules that are compiled and executed independently to (i) handle the simulation loop, (ii) calculate the simulated robot state (eqs. (1)–(3)) or (iii) command a real robot to a new state, (iv) calculate a control command $\mathbf{u}(\mathbf{x}_k, t_k)$, among other.

III. MOBILE ROBOT SIMULATOR REVIEW

This section reviews the simulators in terms of their main features including, architecture aspects, flexibility and modularity issues. Also some comments concerning the installation, usage, and drawbacks of each of these simulation tools are included in this section. For step-by-step installation and usage notes we have created a site that includes this information [38] from which also the source code of the simulations presented in this paper can be obtained. A comparison summary of the tools and their most relevant features is presented in table I. The best simulators found are discussed in sections III-A through III-C, while the remaining simulators are mentioned in section III-D.

A. Carmen: Carnegie Mellon Robot Navigation Toolkit

Carmen [14] is collection of open-source software for mobile robot control, which provides basic navigation functions for base control, sensor reading and logging, localization, path planning, mapping and obstacle avoidance. Carmen was developed by researchers at Carnegie Mellon's Robotics Institute and sponsored by DARPA's MARS Program. The development of Carmen's first beta 0.1 versions dates back to October 2004. Version 0.6 was released on SourceForge in April 2006. The current version is beta 0.7.4 and was released on October 2008.

Among Carmen's core functionalities are (i) support for different mobile robot platforms: iRobot's ATRV, B21R, ActivMedia Pioneer robots, Nomadic Technologies Scout and XR4000 and Segway, (ii) support for various sensors: Sick LMS and PLS laser scanners, Hokuyo IR PB9 obstacle detection sensor, NMEA GPS data streams, (iii) simulation in 2D only, (iv) navigation algorithms for localization, path planning, mapping, (v) written in C language with some support for Java; currently it runs under Linux only, highly modular design relying on the well-established and supported Inter Process Communication (IPC) library developed since 1994 at CMU by Reid Simmons. Using the IPC library, a central server application handles all messages from the client modules as shown in fig. 1. The modules include the robot base interface, the sensors interface, a control module a graphical user-interface (optional) and the simulator module. Each module subscribes to certain messages, which are managed by the central server. The modules may be queried by the central server, which then publishes the messages to the subscribing modules. This architecture can be slower than using a shared-memory approach, but allows to run the modules on different host machines over a network. This is a powerful feature because of two reasons. First, it allows to control or simulate systems with multiple agents. Second, in many real systems some navigation tasks may require a dedicated processor due to their complexity, thus making it necessary to distribute the whole process over a network.

The installation of Carmen is managed through configure and Makefile scripts, however, due to changes in Linux and its libraries, several packages must be installed manually and some modifications to the installation scripts are necessary. This fact when combined with Carmen's incomplete documentation make installation a tortuous process, which can be far from trivial to non-expert Linux user's. We have provided full instructions in order to successfully install Carmen 0.7.4 on Linux Fedora 12 in [38]. Although the Carmen's website provides some notes on how to quickly get Carmen running, these notes are only useful to run the code distributed as is. Additional documentation with examples on how to add new modules or program navigation strategies is not included on the project's website. Therefore, creating new components and adding algorithms requires the user to do some reverse engineering to find out from the source code what the system is doing. We also provide an example in [38] about how to modify the original Carmen project files in order to include a new robot controller and carry out simulations. The example is based on the simulation presented later in this paper.

A major drawback of Carmen when compared to PSG or ODE is the early stage of development of its documentation, which makes it difficult to people who have not been directly involved with the Carmen's development to be familiar with the proper programming an user practices. Another aspect which seems to limit the applicability of Carmen to a wide range of problems is the fact that the simulator only considers 2D kinematic models. A considerable development

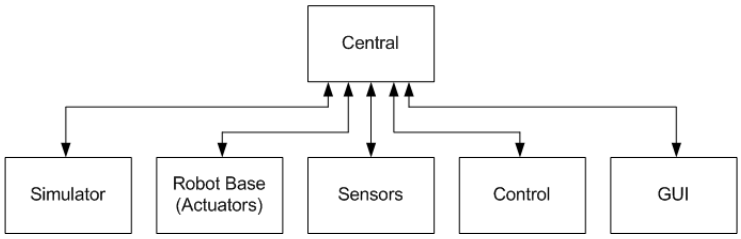

Fig. 1. Carmen IPC architecture based on TCP sockets for communication between several processes and a central server.

effort would be necessary to endow Carmen with additional functionality for 3D simulation. It is fair to say that Carmen is well thought, compact, and its current distribution is clean. With improvements in its documentation it should become a great tool for testing novel control and simultaneous localization and mapping algorithms in 2D environments with possibly multiple robots.

B. Player-Stage-Gazebo (PSG)

PSG is an open-source project available in Sourceforge [16], which consists of the Stage (2D) and Gazebo (3D) simulators and the Player network server. Application programmers may use different Application Programmer Interfaces (APIs) and libraries depending on whether they want to connect to Player, Stage or Gazebo and what language they want to use. The PSG project aims for POSIX compliance and runs on most UNIX-like OS's. As previously mentioned, Stage is a 2D simulator and using Player is only practical for non real-time experiments with slow-moving, statically stable wheeled robot as Player was designed to support update rates of the order of 5-100Hz [17], [18]. Because of these issues, we have focused our current investigation on the Gazebo 3D simulator.

If a user requires better real-time support, they can connect to the Gazebo (currently at version 0.10.0) simulator directly by using libgazebo. This library connects to the simulator using file based shared memory which is a fast and reliable communication mechanism that does not have some of the severe time constraints of TCP/IP.

Gazebo and Stage do not simulate specific devices (such as a SICK LMS-200 or Pioneer robot), instead they use abstract models (such as "laser" and "position"). These models are defined and configured in a world file. Available sensor models included with Gazebo include: sonar, scanning laser range-finders, GPS and Inertial Measurement System (IMU), monocular and stereo cameras. Robots models available include most commonly used robot types such as: Pioneer2DX, Pioneer2AT and SegwayRMP.

As seen in fig. 2, Gazebo is implemented as a standalone simulator (i.e. Gazebo Simulator) and the interface library (i.e. libgazebo) [16] that can be used without Player and its library (e.g. libplayerc++).

Unfortunately Gazebo is notoriously difficult to install and run due to its complexity [17]. In order to function properly, Gazebo has many non documented dependencies that include specific versions of the third party libraries. For example, we found that the latest unnumbered version of Gazebo as well

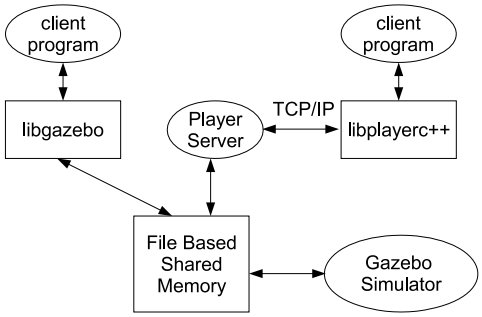

Fig. 2. PSG Architecture

as the latest released version (i.e. 0.10.0) would not properly update shared memory when OGRE 1.7.0 was used. Version 1.6.4 of OGRE was required for proper functionality of the system. The list of commands in order to install Gazebo is very long to be included here, we have provided full instructions in order to successfully install Gazebo 0.10.0 on Ubuntu 9.10 in [38].

The libgazebo API is fairly easy to use and understand, a libgazebo application is implemented as code written in C++ that includes the library "gazebo.h". The interaction with the simulation requires interface objects that first must be opened to be used for various tasks such as allowing the user to dynamically create models (e.g. gazebo::Factory), reading/setting simulation data (e.g. gazebo::SimulationIface), reading sensor data (e.g. gazebo::RaySensor) and sending commands to the robot(e.g. gazebo::PositionIface). Gazebo includes the graphics renderer OGRE by default and the simulation output is automatically rendered visually.

C. Open Dynamics Engine (ODE)

ODE [19] is an open-source physics engine that has been under development by Russell Smith since 2001. It's a library for simulating articulated rigid body dynamics and has a built-in collision detection. ODE is used for simulating dynamic interactions between bodies and is not tied to any particular graphics package. ODE has already been used in many applications and games, such as OpenSimulator, Player Project, Webots, World of Goo and BloodRayne 2. The last numbered release of ODE is version 0.11.1 which was released in october 2009. ODE features a simulation of articulated rigid-body features, interactive or real-time simulation, highly stable integrator, hard contacts and a built-in collision detection system.

ODE works through the interaction of four essential kind of structures: World, Space, Body and Geom. The structure "World" manages gravity and performs time integration. The

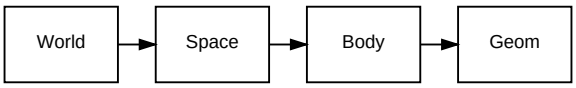

Fig. 3. ODE's basic structures

structure named "Space" is contained in the World and optimizes collision detection. The structures of type "Body" are contained in the World and have physical features. The structures "Geom" define collision detection and rendering of the Body. The source code is provided with the installation and allows to customize ODE from low-level.

One of the main advantages of ODE is that it does not have specific software dependencies. The installation package provides different build systems for each platform and includes ready-to-use workspaces for Visual Studio and Code::Blocks. In order to install ODE 0.11.1 on Ubuntu 9.10 and WinXP the reader may refer to [38].

The API is straightforward, flexible and intuitive. An ODE application is basically implemented a code written in C or C++ that includes the library "ode.h" and usually requires a graphics render. ODE includes the basic graphics render "drawstuff". The documentation [20] and community support is extensive and the project is in active development.

One drawback with ODE, is that in the collision between triangle data structures "trimesh" determining the contact points is not easily implementable and thus primitive collisions are still preferable. Another issue is that as a low-level physics engine, it can be integrated with different projects, but for users who need to develop complex worlds with ease, ODE has no built-in higher level features. Also, ODE emphasizes speed and stability over physical accuracy. This is useful for applications like 3-D games, but inconvenient for applications that require high physical accuracy.

D. Other Simulators

Breve [31] is a package which allows the creation of 3D simulations of multi-agent systems and artificial life using Python scripts. Breve employs OpenGL to generate visually realistic simulations and includes experimental support for articulated body physical simulation and collision detection with static and dynamic friction. Breve's emphasis is placed on visual realistm rather than physically accurate simulations, and the physics engine is a work-in-progress [11].

OpenRave [32] is focused on autonomous motion planning and high-level scripting rather than low-level control and message protocols. To this end, OpenRave provides a plugin-based framework which integrates programming interfaces to services, such as, physics, collision detection, kinematics, robot control, based on well established packages like the Bullet Physics Package and the Open Dynamics Engine for physics simulation, the Robot Operating System (ROS), as well as its own modules, which include kinematic models for various types of robots, sensor models and motion planners. OpenRave is a promising project still under active development [12]. A drawback of this tool is that getting started may not be very simple, since can require some knowledge of different external packages that are involved to achieve OpenRave's full functionality.

OpenSim [33] is based on Open Scene Graph and Demeter (OpenGL) for rendering, and the Open Dynamics Engine (ODE) as physics engine. However, its development is no longer active since 2006.

Simbad [34] is a 3D robot simulator for research and education purposes. It is intended for people who want to study machine learning and artificial intelligence algorithms in the context of autonomous agents. It emphasizes readability, simplicity and user-friendliness over accurate real world simulations, and therefore, it only utilizes a simplified physics model [13].

UsarSim [35] (Unified System for Automation and Robot Simulation) is a high-fidelity robot and environment simulation tool based on the Unreal Tournament game engine. Because it is based on a powerful and well-known game engine, it promises to become a stable platform for realistic world simulations at the macroscopic level. This means however that at a finer level of detail, this simulator may not be used in its current stage for accurately modeling a robot's internal forces or the interaction forces between the robot and its environment. Nonetheless, UsarSim is used in the RoboCup Rescue Virtual Robots Competition and the IEEE Virtual Manufacturing and Automation Competition. UsarSim relies on the Player Project [16] as network server to provide a unified interface for controlling the robots and communicating with sensors and actuators. UsarSim is still at a beta stage of development, and may not be very user friendly yet as it depends on several external packages. Therefore, the user may find it a daunting task to become familiarized with all the documentation of external components in addition to UsarSim's own documentation.

There exist other mobile robot simulators, such as EyeSim [36] (for the EyeRobot) and RoboCode [37] (for virtual robot tank battles). However, these simulators are conceived to specifically simulate one type of robot under some limited scenarios, and therefore cannot be considered completely versatile simulation libraries or packages.

IV. SIMULATION RESULTS

A. Testing Methodology

Testing of the different simulators was done using the same model parameters or as similar as possible when the simulator did not include an equivalent configuration option. The simulated world is an area of at least 5×5 m. The robot model employed is that of an MobileRobots Pioneer 2. Its dimensions are approximated to $0.45 \times 0.30 \times 0.20$ m (length, width, and height). The distance between the wheels is 0.4 m and the wheels' diameter is 0.15 m. The robot's body weight is 21.5 kg. The robot was programmed to perform a 3×3 m square-shaped trajectory under open-loop control. To this end, the translation velocity was set to 0.1 m/s and the rotation velocity was set to $\omega = (v_R - v_L)/d_w$, where $v_R = -v_L = 0.1$ m/s are the right and left wheel translational velocities, and $d_w = 0.4$ m is the distance between wheels, i.e. $\omega = 0.5$ rad/s. In each case the robot was instructed to run for a set amount of time until it reached

the first corner of the square shape, then it performed a rotation for a set amount of time and then it was instructed to run to the next corner and so on. For each robot we performed 30 runs to compute performance statistics and obtain motion trajectories shown in fig. 4. The results are discussed and compared in the next section.

B. Results Comparison

The simulators are compared in terms of the trajectory accuracy, the simulation precision, and the computation time, which is related to the CPU usage. The trajectory accuracy is measured in terms of the end-point error. When traversing a square path, cumulative simulation errors become evident due to two facts. First, the rotation angles which should be 90° have some error, which causes the square not to be perfectly square. Second, the traversed distance along each side of the square is not exactly 3 m. Moreover, when the robot moves in one direction, the length of the path is not the same as that when the robot moves in the opposite direction, thus resulting in a final position which does not exactly match the starting point. The larger the cumulative errors in translation and rotation, the larger the error between the final and initial points of the trajectories. Therefore the motion model and simulator accuracy can be assumed to be proportional to the end point error and simulation precision can be computed directly from the standard deviation of the simulated trajectories end-points.

The results obtained show that Carmen has the largest end-point position error (29 cm), followed by Gazebo (24 cm) and ODE (21 cm), as shown in table II. In terms of precision, ODE is also the best simulator, yielding negligible standard deviation values of the order of 10^{-9} or smaller for the final position. The precision of Carmen is better than 1 cm (0.3%, if computed over 3 m, the length of the square's side). Gazebo exhibits the worst precision, with a standard deviation of the resulting end-point of 11 and 17 cm for x and y respectively (a relative error of almost 0.7%), these results were verified multiple times and on different machines. In terms of CPU usage, Carmen is more intensive because it uses a TCP socket based communication scheme between the different processs, as opposed to Gazebo's shared memory approach and ODE's implementation which results in a single executable file after compilation.

V. FINAL REMARKS

This paper presented a survey of the existing software tools for mobile robot simulation in section III. Introductory background aspects about simulation processes and simulator architectures were discussed in section II. This information can be of tutorial value to researchers and developers interested in the simulation of robotic systems.

Three top open-source simulators were selected for detailed comparison considering their level of maturity, modularity, and to some extent, their popularity: Carmen, Player-Stage-Gazebo (PSG) and the Open Dynamics Engine (ODE). Despite being still in beta stages of development, it is possible to state that ODE is perhaps the most documented and easy to install and learn. ODE is very flexible as its primary objective is to serve as a physics engine, but it also includes some visualization components to achieve 3D views of the simulation. A disadvantage is that it requires developing from scratch code specific to robot navigation. However, there are many projects relying on ODE which include some ready-made mechanisms, sensor and actuator models that can be reused. On the other hand, Carmen and PSG provide a framework conceived specifically for simulation of mobile robots in multi-agent scenarios. The major differences between Carmen and PSG are in the fact that PSG includes 3D dynamical models, while Carmen considers only 2D kinematic models in its simulation. The results section IV shows that ODE is the most accurate simulator, followed by Carmen and then PSG. This can be explained because ODE is specifically conceived for physics simulation and has a monolithic structure, unlike Carmen or PSG which run different process that must communicate with certain latencies. The latter is not necessarily a disadvantage, since a real hardware implementation may also require a distributed process approach. The reasons for the differences in simulation precision between Carmen and PSG have not been fully established. Further research will be necessary to determine if this is a problem due to differences in the simulator's architecture (i.e. using shared memory vs. TCP sockets) or some other internal implementation aspect.

Both Carmen and Gazebo in their current distributions are difficult to install because of missing libraries and other details that can be fixed if the user has sufficient knowledge of Linux operating systems. Documentation seems to be rather incomplete, hence both still require more work to become user-friendly packages, ready to use without trying to reverse engineer the code to figure out why certain things do not work right the first time. We created a site [38] with install notes and simulation examples to help new users of these tools to hopefully get their systems up and running more quickly, and with much less trouble.

REFERENCES

[1] L.E. Chiang. Teaching Robotics with a Reconfigurable 3D Multibody Dynamics Simulator. *Computer Applications in Engineering Education*, v. 18, n. 1, pp. 108–116, 12 Feb. 2009.

[2] L. Žlajpah. Simulation in Robotics. *Mathematics and Computers in Simulation*, v. 79, n. 4, pp. 879–897, 15 Dec. 2008.

[3] L. Žlajpah. Integrated Environment for Modelling, Simulation and Control Design for Robotic Manipulators. *Journal of Intelligent and Robotic Systems*, v. 32, n. 2, pp. 219–234, Oct. 2001.

[4] Z. Kovačić, S. Bogdan, K. Petrinec, T. Reichenbach, M. Punčec. Leonardo – The Off-line Programming Tool for Robotized Plants. *Proceedings of the 9^{th} Mediterranean Conference on Control and Automation*, Dubrovnik, Croatia, June 27-29, 2001.

[5] A. Miller and P.K. Allen. Graspit!: A Versatile Simulator for Robotic Grasping. *IEEE Robotics and Automation Magazine*, v. 11, n. 4, pp. 110–122, Dec. 2004.

[6] J. Wang, C. Gosselin and L. Cheng. Modeling and Simulation of Robotic Systems with Closed Kinematic Chains Using the Virtual Spring Approach. *Multibody System Dynamics*, v. 7, pp. 145–170, 2002.

[7] T. Reichenbach. A dynamic simulator for humanoid robots. *Artificial Life and Robotics*, v. 13, n. 2, pp. 561–565, March 2009.

[8] D.V. Marhefka and D.E. Orin. XAnimate: An Educational Tool for Robot Graphical Simulation. *IEEE Robotics and Automation Magazine*, v. 3, n. 2, pp. 6–14, June 1996.

TABLE I

SUMMARY OF ROBOT SIMULATION TOOLS.

Software	Dynamic Configuration	Installation Complexity (LOW,MED,HIGH)	Manipulators	Mobile Platforms	2D	3D	Kinematics	Dynamics	Modular	Open	Platform	Target Applications
Carmen	–	HIGH	–	✓	✓	–	✓	–	✓	✓	Linux	Robot interfacing, localization and navigation.
Gazebo	✓	HIGH	✓	✓	✓	✓	✓	✓	✓	✓	Linux	Education, visualization, localization and navigation.
ODE	✓	LOW	✓	✓	✓	✓	✓	✓	✓	✓	Linux/Windows	Engine for general rigid body dynamics simulation.

TABLE II

SIMULATION RESULTS.

Software	Endpoint Position RMS Error [m]	Endpoint Std. Dev σ_x	σ_y	σ_θ	Processing Time [s]
Carmen	0.29	0.0089	0.0115	3×10^{-17}	141.79
Gazebo	0.2407	0.1142	0.1719	2.3616	71.4667
ODE	0.2079	5.6836×10^{-9}	7.0×10^{-15}	3.0×10^{-15}	81.3

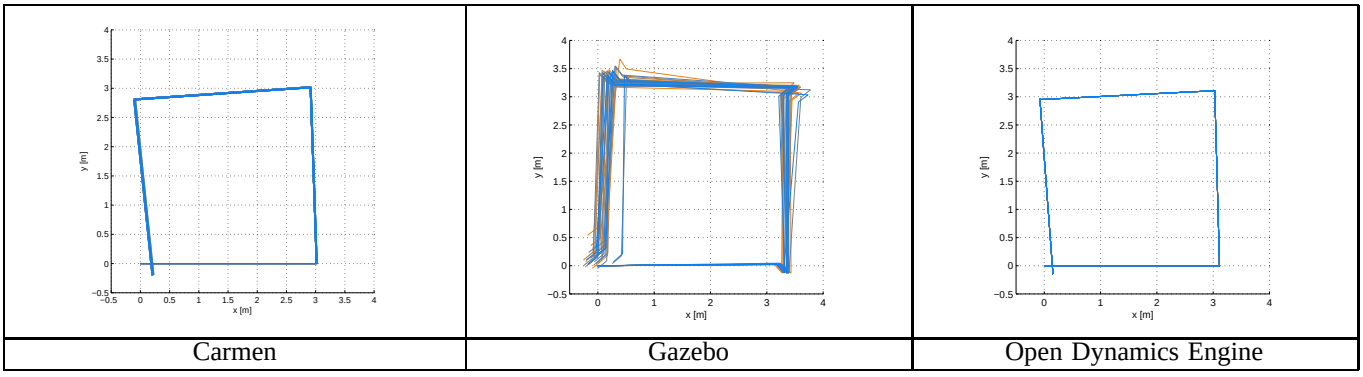

Fig. 4. Comparison of 2D trajectories.

[9] P.I. Corke. A Robotics Toolbox for MATLAB. *IEEE Robotics and Automation Magazine*, v. 3, n. 1, pp. 24–32, Mar. 1996.

[10] J. Nethery, M. Spong. Robotica: A Mathematica Package for Robot Analysis. *IEEE Robotics and Automation Magazine*, v. 1, n. 1, pp. 13–20, Jan. 1994.

[11] J. Klein. breve: A 3D Simulation Environment for the Simulation of Decentralized Systems and Artificial Life. *Proceedings of Artificial Life VIII, the 8th International Conference on the Simulation and Synthesis of Living Systems*. The MIT Press, 2002.

[12] R. Diankov and J. Kuffner. OpenRAVE: A Planning Architecture for Autonomous Robotics. Tech. Report CMU-RI-TR-08-34, Robotics Institute, Carnegie Mellon University, July 2008.

[13] L. Hugues and N. Bredeche. Simbad: An Autonomous Robot Simulation Package for Education and Research. *Simulation of Adaptive Behavior (SAB 2006)*, Rome, Italy, 2006.

[14] Carmen: Carnegie Mellon Robot Navigation Toolkit. `http://carmen.sourceforge.net/`

[15] W.R. Stevens. *Unix Network Programming, Volume 2, Interprocess Communications*, 2^{nd} ed., Prentice-Hall, Inc., 1999.

[16] Gazebo: The Player Project, Free Software Tools for Robot and Sensor Applications. `http://playerstage.sourceforge.net/`

[17] R.T. Vaughan and B.P. Gerkey, *Reusable robot code and the player/stage project*, Springer-Verlag, Berlin, pp. 267–289 2006.

[18] E. Folgado, M. Rincon, J.R. Alvarez and J. Mira. A Multi-robot Surveillance System Simulated in Gazebo. *Lecture Notes in Computer Science*, v. 4528, n. 4, pp. 202–211, June. 2007.

[19] ODE: Open Dynamics Engine. `http://www.ode.org/`

[20] ODE Manual. `http://opende.sourceforge.net/wiki/index.php/Manual`

[21] A. Rahim, J. Teo and A. Saudi. An Empirical Comparison of Code Size Limit in Auto-Constructive Artificial Life. *IEEE Conference on Cybernetics and Intelligent Systems*, pp. 1–6, June 2006.

[22] R.B. Rusu, A. Holzbach, R. Diankov, G. Bradski and M. Beetz. Perception for Mobile Manipulation and Grasping Using Active Stereo. *9th IEEE-RAS International Conference on Humanoid Robots*, pp. 632–638, December 2009.

[23] S.L. Delp, F.C. Anderson, A.S. Arnold, P. Loan, A. Habib, C.T. John, E. Guendelman and D.G. Thelen. OpenSim: Open-Source Software to Create and Analyze Dynamic Simulations of Movement. *IEEE Transactions on Biomedical Engineering*, v. 54, n. 11, pp. 1940–1950, November 2007.

[24] K. Kobayashi, Y. Uchida and K. Watanabe. A study of battle strategy for the Robocode. *SICE Annual Conference*, v. 3, pp. 3373–3376, August 2003.

[25] S. Carpin, M. Lewis, J. Wang, S. Balakirsky and C. Scrapper. USARSim: a robot simulator for research and education, *IEEE International Conference on Robotics and Automation*, pp. 1400–1405, April 2007.

[26] I.G. Daza, L.M.B. Pascual, M.A.S. Vázquez, E.L. Guillen, R.B. Navarro and L.B. Vázquez. Low Level Control in States Space for the Pioneer. *The International Conference on Computer as a Tool*, v. 1, pp. 322–325, November 2005.

[27] Cogmation Robotics. `http://www.cogmation.com/`

[28] anyKode Marilou. `http://www.anykode.com/`

[29] J. Prevost, M.A. Joordens and M. Jamshidi. Simulation of Underwater Robots using MS Robot Studio. *IEEE International Conference on System of Systems Engineering*, pp. 1–5, June 2008.

[30] B. Magyar, Z. Forhecz and P. Korondi. Developing an Efficient Mobile Robot Control Algorithm in the Webots Simulation Environment. *IEEE International Conference on Industrial Technology*, v. 1, pp. 179–184, December 2003

[31] The Breve simulator. `http://www.spiderland.org/`

[32] OpenRAVE. `http://openrave.programmingvision.com/`

[33] OpenSim. `http://opensimulator.sourceforge.net/`

[34] Simbad. `http://simbad.sourceforge.net/`

[35] USARSim. `http://sourceforge.net/projects/usarsim/`

[36] EyeSim. `http://robotics.ee.uwa.edu.au/eyebot/doc/sim/sim.html`

[37] Robocode. `http://robocode.sourceforge.net/`

[38] Installation istructions for Carmen, Gazebo and Open Dynamics Engine. `https://sourceforge.net/apps/mediawiki/arsproject/`