On Open- source Multi-robot simulators

Molaletsa Namoshe[1], N S Tlale[1], C M Kumile[2]

[1]*Department of Material Science and Manufacturing Centre for Scientific and Industrial Research, P.O. Box 395, Pretoria, Republic of South Africa , +27 12 841 3107, +27 12 841 3700,* mnamoshe@csir.co.za
+27 12 841 4395, ntlale@csir.co.za

[2]*Department of Mechanical Engineering, Tshwane University of Technology, Private bag X680, +27 12 382 5114, +27 12 382 5286, Pretoria, South Africa*
kumilecm@tut.ac.za

Abstract-Open source software simulators play a major role in robotics design and research as platforms for developing, testing and improving architectures, concepts and algorithms for cooperative/multi-robot systems. Simulation environment enables control systems to be developed rapidly and transferred to the real system with minimal change in behaviour. A number of simulators boost of effective data flow between sensors, processors and actuators of a robot/ team through a communication channel. The drive to build an adaptable and capable robotic development environment has led to the emergence of numerous open source application programs. This article discusses freely available open source simulators and their impact in multi mobile robot system research projects. Firstly, the architectures of the different simulation software are discussed. Secondly, their capabilities are discussed with regards to their architectures. Thirdly, multi-robot cooperation theory is presented. Lastly, an example of algorithm development for multi-robot co-operation using one of the discussed simulation platforms is presented.

Key Words: Open source, multi robot system, simulators, robotic environment

I Introduction

Despite advances in recent years, design and implementation of control architectures for multi robot systems still remain a challenge. The difficulty stems from the fact that autonomous mobile robots incorporate quite diverse sensory and actuatory devices which need to be controlled simultaneously by a group of networked computers. These devices, form part of components embodied in to a complete unit, an autonomous mobile robot. A robot needs to reason at a high level of abstraction to choose appropriate behaviour given a particular situation. In most cases these behaviours are carried out in a sequence of coordinated events, leading to increased complexity. The sheer size of work at hand, as well as unavailability of resources makes the development, testing and analysis of control algorithms on physical robots an expensive process. Therefore, as a tool for enhancing the understanding of robot control systems, a number of open source simulators were developed. Simulators offer a number of important applications: it enables developers to focus solely on developing and testing control programs with little regard to the hardware aspects of robotic platforms. This is a very important aspect in mobile robotics in cases where a group of people are involved on different parts of a single project. In addition, simulators offer a programmer access to data which is not easily attainable with real robots. These aspects have lead to a number of institutions around the world adopting simulators as teaching aids.

Although simulation environment offers the aforementioned advantages, experiment on a physical system must also be done because simulators provide a simplified model of the real robot and its surrounding. The reason being, a simulator cannot completely capture the complexity of a robot and the level of noise found in physical world. That is, Physical aspects like backlash in gears and lags in response time are difficult to characterize and accurately model.

In light of all these limitations however, open source simulators offers a safe, cost-effective alternative to real environments. Therefore, in order to develop reliable and robust control algorithms for autonomous robots it is important to use simulators with medium to high physical fidelity [1]. An algorithm developed in a favourable platform fidelity can be implemented on a real robot with minimal or no change to the algorithm.

This paper discusses open source simulators and their capabilities in multi robot systems. In order for a simulator to realize a multi robot control system, the underlying middleware must support distributed computing paradigm. Where a middleware is a software platform that uses standardized interfaces and protocols to provide services for the communication of distributed computer architectures [2].

This paper is organised as follows: section 2 gives a brief discussion about related open simulation environment. The third section discusses about five simulators in use in labs around the world and their underlying software to implement distributed computing. Criteria used to select stage simulator in this paper is covered in section 4 while section 5 gives an simple exploration example carried out using Player/ Stage.

II. Related work

There are a number of open source simulators used for robot development around the world. The main benefits to the programmer in using a simulator over a real robot are convenience and cost. To minimize already high cost in robotics, many robotic communities opt for freely available software simulators. Commercial simulators like WEBOTS [21] offer complex high fidelity environment but this paper focuses on free and available simulators for public modification. Related simulation environment include, Carmen (Carnegie Mellon Robot Navigation Toolkit) [20] an open-source

collection of software for mobile robot control. CARMEN is modular software supported by IPC (Inter-Process Communication System) and provides basic navigation primitives. The simulator was designed for single robot control but it is possible to implement multi robot as well. Darwin 2K [19] was created by Chris Leger at Carnegie Mellon University and is mainly tailored for evolutionary robotics. The simulator correctly models motor and gear heads as well as stress estimates on structural bodies. OpenSim [18] is a 3D simulator based on ODE (Open Dynamics engine) physics engine [5] and is under development by David Jung.

III. Open Source Simulators

A. SimRobot

SimRobot [3], [4] is a robotic simulation environment that enables users to easily implement robot simulation models in a 3D space. SimRobot is platform impendent, i.e. a large number of body elements and robot device models offer programmers a leeway to randomly combine these components to form an arbitrary robot model. The simulator uses ODE to simulate rigid body dynamics while visualisation and computation of imagery sensor data is based on OpenGL [13]. Configuration of robot models as well their position state is carried out by XML-based modelling language. Generic devices supported by the SimRobot include: cameras, range sensors, touch sensors, and actuator state.

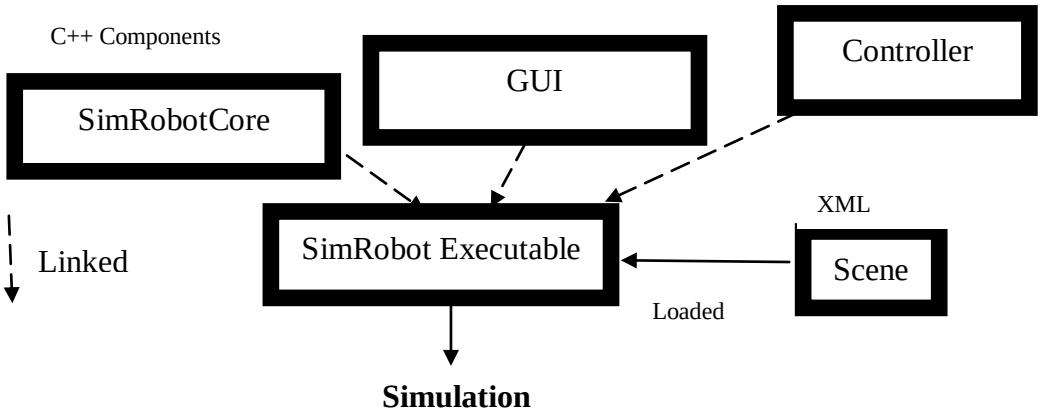


Figure 1. Modular based SimRobot Architecture

SimRobot architecture depicted in figure 1 above shows a modular/ components based architecture where modules are linked to a single application, i.e. SimRobot Executable. The main idea behind the architecture is that it offers the best result in progressive execution of the whole software even if there is no concurrence. The approach differs from other Client/ server architecture because it offers a programmer a step by step execution of the software. SimRobotCore is the engine/ kernel of the simulator. It models both robot and the environment and simulates sensor reading as well as executing control commands supplied by clients or graphical user interface module. The highlight of SimRobot in multi-robot system set up was its successful implementation by the German team in their 2005 RoboCup team. Halting

B. Microsoft Robotics Studio

Microsoft Robotics Studio [6] is a freely available Windows-based environment for robot control and simulation. The robotic Studio software enables developers to create applications for a diverse variety of robotic hardware i.e. the runtime is designed to support any type of robotic application. These applications are realised by using Microsoft Visual Programming Language (MVPL) which is a development environment designed on a graphical data flow-based programming model rather than control flow. Additionally, a collection of these linked components can be treated as a single block to be networked with other blocks during robot application building, i.e. reusability of components. Consequently, Visual Programming Language is suitable for programming concurrent or distributed systems. The studio archive distributed service architecture by running on a windows computer at the same time execute on platforms housing microcontrollers as well as running a simulator. Microsoft Visual Simulation Environment is based on PhysX(tm) engine from AGEIA(tm), a pioneer in hardware-accelerated physics, enabling real-world physics simulations with robot models [7].

The Studio Runtime supports a variety of windows platforms supported by .NET framework as well as WinCE using .NET compact framework. It is made up of two components, i.e. The Concurrency and Coordination Runtime (CCR) and Decentralized software Services (DSS) mainly to develop robotic applications. CCR is service-oriented applications software designed to handle concurrent and asynchronous operations. This application model allows a user to design loosely coupled software components which interact through messages, facilitating heterogeneous hardware integration and networked applications. DSS module is a hosting environment that provides support for the creation and management of services. Services in MS Studio include software and hardware components mainly networked and inter-communicating within a DSS node or across the network.

C. Sinbad

Sinbad [8] is an open source 3D java-based robot simulator for scientific research and learning. Like a host of other open source simulators, Simbad is freely available for public use and modification under the confines of GPL (GNU General Public Licence). The simulator is characterized as a low fidelity but complex 3D scene modelling using built in simulation engine (Simbad) and provides a collection of tools for Evolutionary Robotics [9]. The main design focus of this simulator is to offer a simple environment to study AI algorithms and machine learning in autonomous mobile robotics set up. Simbad simulator has three main features:
The simulator **-**As aforementioned Simbad is a 3d java-based multi-robot simulator tailored for scientific research as well as educational purposes. The simulator runs on any platform which houses the standard Java development Kit and Sun Java 3d components, e.g. Windows, Mac and Linux.
PicoNode- It is a graph based robot control representation framework based on neural network architecture and it is shipped with two implementations: the feedforwad and recurrent neural networks. The feature offer robot programmers a simple way to build multi-layered perceptron as well as N-layers recurrent nets in neural networks paradigm.
PicoEvo- is a general library of evolutionary Algorithms which include Genetic Algorithms, Evolutionary strategies, tree-based and graph based programming.
Simbad has low fidelity physics engine, meaning robot controllers build in this environment need to be changed considerable in order to be tested or implemented on real robot.

D. MissionLab

MissionLab [10] is a set of software tools for the development and testing of robotic behaviour either for single or in a team of cooperating/ collaborating robot. It supports implementation of both virtual and real robotic platforms as well as device drivers for controlling iRobot's ATRV-Jr and Urban Robot, ActivMedia's AmigoBot and Pioneer AT, and Nomadics Technologies' Nomad 150 & 200. It is oriented towards reconnaissance missions by a robot team. Each robot executes its part of mission using reactive control methods developed by Georgia Institute of Technology. The software controller is distributed, i.e. the user's console could be run on one computer while multiple robot control executables are distributed across the network. MissionLab relies on IPT, InterProcess communications Toolkit from Carnegie Mellon University [11], for communication between the robots and mlab, the user's console. It offer socket based communication using TCP/IP and supports both publish/ Subscribe and client/ server protocols.
An operator explicitly states robotic missions (graphically) by finite state diagrams using the CfgEdit (Configuration Editor). A Library of robot behaviours by system developers offers the user numerous options to configure these behaviours for a robot/ team to perform a particular task.
Compiling causes MissionLab to generate a chain of binary executable. The simulation first compiles Configuration Description Language (CDL), which is a representation of robotic behaviours into the Configuration Network Language (CNL) code. CNL code is then compiled into C++ code and ultimately machine code. The resulting robot program contains a communication unit (HClient) to interface with the hardware server (HServer). This sever provide an abstract control interface to a number of robot devices via IPT communication software.

E. Player/ Stage project

Player/ Stage project [15],[16] is an open source software project tailored for robotic systems research and it provides an infrastructure for distributed access to a number of popular robotic hardware devices. Player and stage run on many UNIX-like platforms, and are released as Free Software under the GNU General Public License and are maintained by the public at http://playerstage.sourceforge.net. This project has got a third member to its family, named Gazebo [14] which is a high fidelity 3D

outdoor dynamics simulator. Stage like Gazebo is a player plugin and it simulates a population of mobile robots, sensors and objects in a 2D bitmapped environment, called worlds. Payer is the core of the project; it is a networked device repository server to popular robotic platforms. The middleware was design in a client server paradigm, and it is a general purpose abstraction layer for defining and controlling robot platforms. In a TCP client/ server transport, devices reside on the server and a control program is a client to the server. The client can run on an onboard computer or any computer that has network connectivity to the robot, i.e. distributed computing. Player supports two kinds of communication mechanism as shown by the figure 2 below, which are client- server communication and Device-Device communication. Inter-device communication is possible within a player server or between servers by passthrough device. Player allows for Inter client communication mechanism but this is left to the programmer to figure it out because it is not documented.

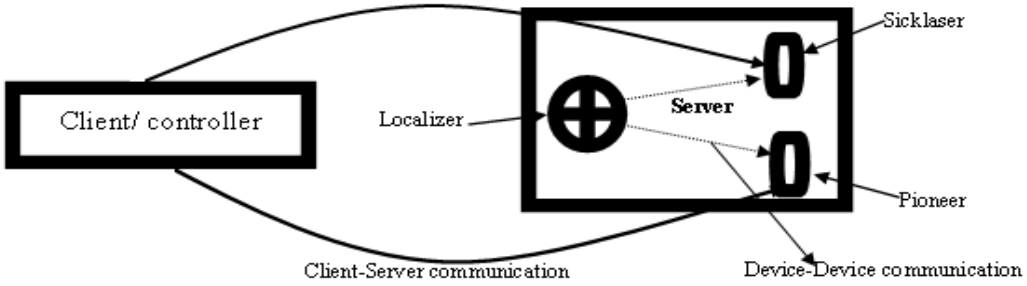


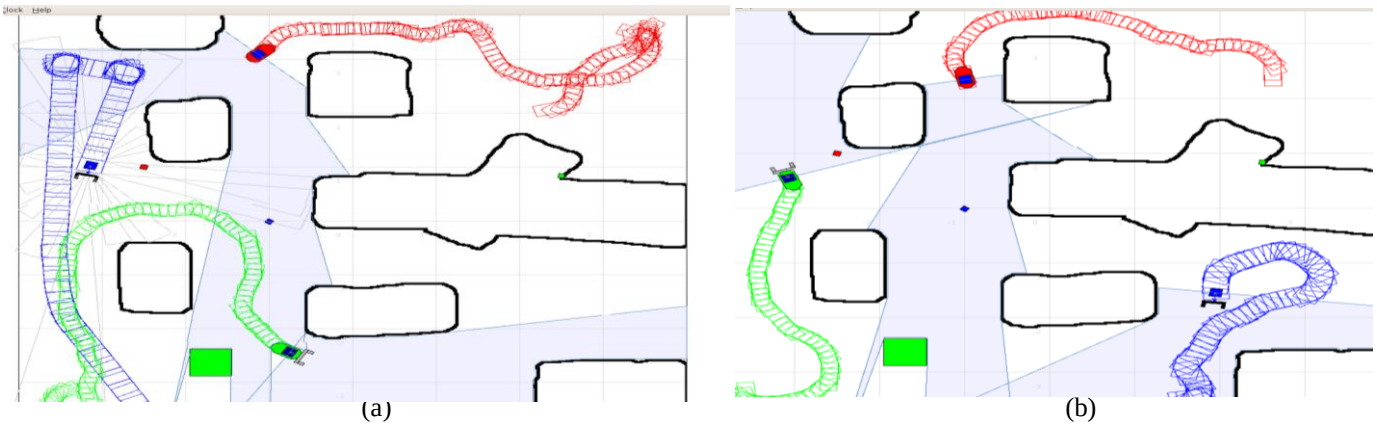
Figure 2. Player architecture

IV. Simulator selection Criteria

Player/ Stage simulator was preferred in this paper to implement multi-robot system simulation because it allows a developer to virtually prototype robots as well as continuous iterative design of control algorithms. Player /Stage has become a de facto standard in open source simulators [17]. In addition, there are quite a number of usability requirements that one looks at when choosing a simulator such as but not restricted to the following: 1.There is a large number of robotic communities actively involved in the development of the project, 2. Modular architecture, 3. Hardware abstraction [12], 4. Platform independent, and 5. Clients side is language independent. 6. Fair physics fidelity simulator to moderately models the world. This is important in the transfer of algorithms from the simulation platform to a real robot as small changes to the algorithm would be required.

V. Behaviour-based multi-robot exploration

The paper present simulation results of multi-robot indoor exploration using behaviour based approach on Stage. This control scheme is achieved through the use of perception-action units, called behaviours. Where each behaviour has an activation level depending on feedback information from carefully selected sensor readings. Behaviours such as *ObstacleAvoidance*, *RandomWonder* and *EscapeEntrapment* and are set a prior. *ObstacleAvoidance* mode is entered when laser or sonar sensor readings fall within a certain threshold. Hence, for every scenario encountered, an appropriate behaviour is executed and commands are sent to fitting actuator to respond accordingly. Maximum spatial coverage by a robot team member is gained by either wondering towards areas where there is maximum laser return or away from other robot or obstacles. Figure 3 below illustrate a simple behaviour based exploration by multi-robot team. Robot models in the simulation use laser or sonar for obstacle avoidance. As mentioned before, stage simulator offer programmers an iterative development platform, as a result a continual improvement to the control scheme will be carried out. The main aim is to develop a fully autonomous control algorithm implying that planning should be added, to form hybrid control architecture.



(a) (b)

Figure 3. Showing simulation results done on stage. In (a) a blue robot, (equipped with sonar) turns away from a red robot and travel to where there is maximum sonar reading return. In (b), robots use laser for object detection.

VII Conclusion and Future Work

This paper explored freely available mobile robot simulators, and looked at their appropriateness to implement multi-robot scenarios. The suitability to realize multi robot system is made possible by the software platforms shipped with these development environments. This has lead to robotic communities around the world to rapidly prototype robotic control algorithms within these platforms, providing an efficient and cost effective alternative to real environments. The resulting control schemes can be transferred from a simulation platform to a physical robot with minimal overhaul. The transfer though is highly dependent on the ability of the simulator to model the physics characteristics of the environment as well as the dynamic nature of multi robot scenes. Therefore, selecting a useful simulator one looks at a number of usability requirements which include ease of use as well its ability to capture the environment physics. Player/ Stage project was selected for use in this paper due to reasons stated in section IV. Player middleware is a distributed device server that allows control of a variety of robotic sensors and actuators. Player middleware offer modular design structure, hence components developed using the software can be incorporated into the design with little change to the existing ones. A hybrid system will be implemented ultimately to make the system fully autonomous; this means that a localizer device will be plugged to the server as shown in figure 2. A robot team that is able to localize and map its environment has a number of applications in modern days. In an indoor set up, a robot group using hybrid system has modules like *GoalSearching*, *WallFollowing*, and *MessagePass* and *DoorTraverse*. A robot team equipped with these behaviours and planning modules is able to explore and map its environment while tracking a point of interest like in search and rescue missions.

Acknowledgement

The author would like to thank the MMM robotic group for their questions and additions. This work was fully supported by Council for Scientific and Industrial Research (CSIR).

References

[1] A. L. Alexander, T. Brunyé, J. Sidman, S. A. Weil, "From gaming to training: A review on fidelity, immersion, presence, and buy-in and their effects on transfer in pc-based simulations and games," [online]. Available:http://www.darwars.com/downloads/DARWARS%2520papers%252012205.pdf, 2005.

[2] Utz, H., Sablatng, S., Enderle, S., and Kraetzschmar, G, "Miro – Middleware for Mobile Robot Applications", *IEEE Transactions on Robotics and Automation*, 18(4):493–497, 2002.

[3] "Simrobot." [Online]. Available: http://www.informatik.uni-bremen.de/simrobot/

[4] Laue, T., Spiess, K., RŐfer, T.: SimRobot , "A General Physical Robot Simulator and its Application in RoboCup" *In RoboCup 2005: Robot Soccer World Cup IX. Lecture Notes in Artificial Intelligence*, 2006.

[5] Smith, R.: Open dynamics engine – ode, www.ode.org, 2005.

[6] Microsoft robotics studio.[Online]. Available: http://msdn.microsoft.com/robotics/

[7] "About AGEIA PhysX" [online]. Available: http://www.ageia.com/physx/index.html/

[8]"Simbad 3d robot simulator." [online]. http://simbad.sourceforge.net/

[9] S.Nolfi & D. Floreano, "Evolutionary Robotics", *MIT press*, 2000.

[10] Balch, T., & Arkin, R, "Behavior-based formation control for multi-robot teams", *IEEE Transactions on Robotics and Automation*, 20(5), 1999.

[11] J. Gowdy, "IPT: An object Oriented Toolkit for InterProces communication", *tech. report CMU-RI-TR-96-07, Robotics Institute, Carnegie Mellon University*, 1996.

[12] R. T. Vaughan, B. P. Gerkey, and A. Howard. "On device abstractions for portable, reusable robot code", *In Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 2003.

[13] "OpenGL" [online]. Available: http://www.opengl.org/

[14] Nathan, K and Howard, A. "Design and use paradigms for Gazebo, an open-source multi-robot simulator", *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, 2004.

[15] Gerkey, B, Howard. A, & Vaughan. R, "Player/Stage. http://playerstage.sourceforge.net", 2005.

[16] Gerkey. B, Vaughan. R, & Howard. A, "The Player/Stage project: Tools for multi-robot and distributed sensor systems", *In Proceedings of the 11th international conference on advanced robotics*, pp. 317–323. Coimbra, Portugal, 2003.

[17] Collett. T. H, Macdonald. A, & Gerkey. B, "Player 2.0: Toward a practical robot programming framework", *In Proc. ACRA'05*, 2005.

[18] OpenSim. [Online]. Available:http://opensimulator.sourceforge.net/.

[19] Darwin2K: Simulation and Automated Synthesis for Robotics. [Online]. Available:http:// Darwin2K.sourceforge.net/.

[20] Montemerlo, M., Roy, N., & Thrun, S. (2003a). CARMEN, Carnegie Mellon Robot Navigation Toolkit. http://carmen.sourceforge.net/.

[21] Webots – http://www.cyberbotics.com/products/webots/