# CoopDynSim: a 3D robotics simulator

Toni Machado, Miguel Sousa, Sérgio Monteiro and Estela Bicho
Department of Industrial Electronics, University of Minho, 4800-058 Guimarães, Portugal
Email: {tmachado, msousa, sergio, estela.bicho}@dei.uminho.pt

***Abstract*—This paper presents *CoopDynSim*, a multi-robot 3D simulator. The main motivations for the development of a new simulation software lie in the need to emulate specific, custom made sensors, combined with the desire to smoothly transfer controller code from simulation to real implementation. The latter is achieved through the use of the same middleware layer already implemented in the real platforms. The high modularity of the solution allows the user to easily add new components or design new platforms. By having independent simulation threads for each robot, distributed control algorithms can easily be tested, abetted by a socket based connection, granting the possibility for an asynchronous, over the network, controller architecture. The ability to run simulations in *real* or *simulated* time, as well as a *play back* option, represent valuable features of the software. The simulator has been used in several projects, with different platforms and distinct control applications, proving it as a heterogeneous and flexible solution. Furthermore, its usage as a teaching tool in a robotics' summer school as well as in an introductory robotics class in our university, upholds its simplicity and user-friendliness.**

## I. Introduction

Computer based robotic simulators have recently gained the attention of researchers [1], [2], [3], [4]. The availability of computers with an ever increasing processing power, combined with accurate physics engines and enhanced visual representations, of both robots and virtual worlds, made simulators go from being component or platform specific, often proprietary with restricted access, to a multi-platform and reconfigurable tool, widely used (especially) in academia.

Simulations provide a mean for one to collect data without the dangers of damaging expensive equipment, otherwise encountered when using the real platforms. For instance, the process of testing a control algorithm or the validation of a new sensor or platform are eased by means of simulation, while keeping the costs low, not only in terms of time spent, but also in terms of human resources needed. Furthermore, it becomes possible to perform tests under specific conditions, which may prove difficult to mimic, or expensive, in the real world, abetted by the availability of multiple (simulated) robots, even if only a few real platforms exist.

With a wide array of simulators accessible nowadays, certain features may help differentiate the available solutions from one another [5], [6], [7]. *Graphical* and *physical accuracy*, as the extent to which the robots and virtual world are similar to their real counterparts, *flexibility*, that is, the type of hardware that can be simulated, and *transparency*, implying the possibility for the user to seamlessly migrate from simulation to the real platforms, represent the key characteristics, from our perspective. Furthermore, the cost of the solution and openness of the source code may be important. Moreover, we argue that the simulator should be simple and user friendly, both in terms of the installation process and normal usage. Our work presents a solution, built from the ground up, which meets the above requirements.

*CoopDynSim* (*Coop*erative *Dyn*amics *Sim*ulator) is built on top of the *Newton Game Dynamics* [8] physics engine, recurring to *OpenGL* [9] to render the environment. Albeit initially designed for the hardware platforms developed in-house, which feature custom made components (difficult to add to other available simulators), it still offers the possibility to add other platforms, designed in third party software, as well as user-defined worlds.

The main strengths of the proposed simulator architecture lie in the modularity and level of abstraction of the robotic components, through the use of a middleware layer. Furthermore, the ability to run in *real* or *simulated* time, as well as a *play back* feature, which allows the user to replay a simulation, represent key characteristics of the solution.

The simulator is being used in several research projects [10], [11] and as a teaching tool for robotics courses in our university, which further help in its validation. *CoopDynSim* is in constant development, and it currently runs on Windows.

The remainder of the paper is organized as follows. Section II presents a brief description of some related work. Section III describes the overall architecture of the simulator, its components and its features. In Section IV, a few use cases are presented. Finally, Sections V and VI conclude the paper and present some guidelines for future work, respectively.

## II. Related work

In this section we briefly describe some of the available simulators. We only aim to give an overall review of a few products, both free and commercially available. For more in-depth surveys on the subject, please refer to [5], [6], [7].

### A. Freeware

One of the most notorious open-source solutions is the *Player-Stage-Gazebo* project [12]. Player represents a hardware network server, and Stage and Gazebo are the 2D and 3D simulators, respectively. Player is a TCP socket based middleware layer, which guarantees abstraction of the robotic hardware modules. Stage is the two-dimensional simulator with low physical accuracy, providing only basic collision detection and simple models. Nonetheless, it excels in the simulation of large groups of robots, such as swarms. Gazebo, on the other hand, is a three-dimensional simulator devised

for simulating a smaller number of platforms. It can make use of the ODE physics engine [13], and multiple sensors and commercial robots are available. The simulator runs only on UNIX based machines and has a challenging installation process, which can represent shortcomings of the solution.

*USARSim* (Unified System for Automation and Robot Simulation) [14], is a simulation tool based on the Unreal engine. Although the simulator itself is free, the engine has a cost associated with it. USARSim is the official simulator used in the RoboCup's Rescue simulation league. Because of its incorporation with the Unreal engine, a high degree of detail and realist world interactions are provided (Karma is the built-in physics engine). Robot's programming and control can be achieved using UnrealScript, but also through other network based frameworks (integration with Player, SIMware and Pyro is possible). USARSim is cross-platform, but its installation process, here also, can be overwhelming, since one has to install the engine and several external packages, as well as become familiar with a large amount of documentation [7].

*Simbad* [15] is an open source simulator written in Java and only requires the Java 3D visualization environment to run, thus making it highly cross-platform. It features only basic physics simulation, being mainly designed for researchers in evolutionary robotics and artificial intelligence, since it includes dedicated libraries for artificial neural networks and genetic algorithms.

*SimRobot* [16], built on top of the ODE physics engine, is mainly used in RoboCup's Standard Platform League. It features a user friendly, drag and drop interface to build the simulated world, as well as the possibility for the user to easily create its own platform, with generic bodies and sensors. One characteristic that distinguishes this solution from others, resides in the built-in code with the simulator's executable, rather than a client/server approach, which the authors argue that allows the user to easily pause and continue the simulation, easing the debugging process.

### B. Commercially available

Within commercial robotics simulators, *Microsoft Robotics Developer Studio* [17] is a popular solution. It is based on the high fidelity physics engine PhysX , and features high quality visualization, with a large collection of robots available. The main programming language used is C#, along with the Visual Programming Language (VPL) developed by Microsoft, which allows users to easily create a control application, without the need of being familiar with programming.

*Webots* [18], developed by Cybertronics, is a multi-platform simulation software that features a large number of commercially available platforms, as well as the possibility for the user to create its own, using any of the existing sensors and actuators. Relying on the ODE physics engine, it is capable of simulating wheeled, legged and flying robots. Programming can be done through C, C++ and Java (TCP connection for external interface is also featured), and applications can be cross compiled for the real hardware platforms.

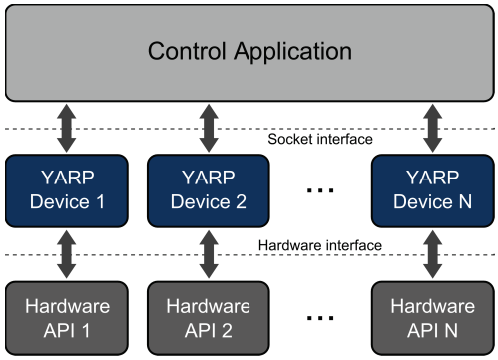


Fig. 1. Schematic of the Hardware Abstraction Layer used in all of our robots.

*robotSim* [19] is another simulation software solution, from Cogmation Robotics, which offers a very realistic environment, along with customizable physics for each robots, which allow the user to tweak how it interacts with the environment, in order to get more realistic simulations. Multiple robots can be simulated, and their (individual) control may be achieved through the provided C++ API or through the network. *robotBuilder* is a concomitant tool from Cogmation Robotics, allowing the user to create a custom platform, with any of the available sensors.

## III. Architecture

*CoopDynSim* is a 3D robotics simulator, developed in C++, capable of emulating multiple robots or teams of robots, obstacles and targets. Newton Game Dynamics [8] is the chosen physics engine, with Open Graphics Library [9] being used to render the scene.

The simulator was developed taking into account the robots existing in our laboratory, i.e. the simulated robots have the same characteristics and interface as the real ones and follow the client-server topology, where each robot is composed of several hardware modules that act as servers, and the control application has clients that connect to each of these modules. This modularized approach makes the addition or removal of hardware (i.e. sensors, actuators) an easy task.

The middleware in use is based on YARP [20] and provides a wrapper with a socket based interface for each of the hardware modules, as illustrated in Fig. 1. Since this abstraction layer is employed both in simulation and in real implementation, the same control application can be used, thus eliminating the, most of the times, hard and time consuming process of migrating from simulation to the real platforms. Nevertheless, a few control parameters may need to be adjusted, due to the fact that the real platforms have unknown perturbations that can not be accounted for in simulation.

To interface with each of the modules, a generic protocol was developed, which is implemented in all of the hardware modules in our laboratory (from robotic manipulators, to vision systems and motor drivers, etc). The message format can be seen in Fig. 2.

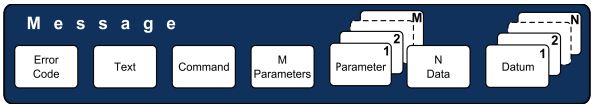


Fig. 2. Communication protocol.

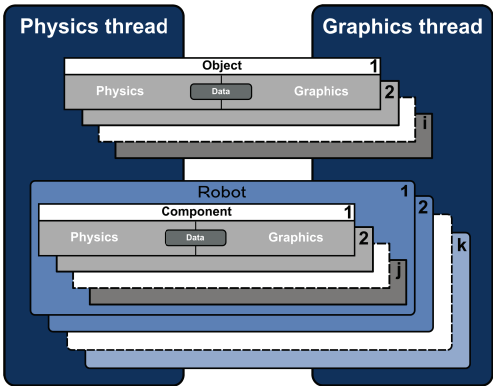


Fig. 3. Thread implementation of the virtual environment.

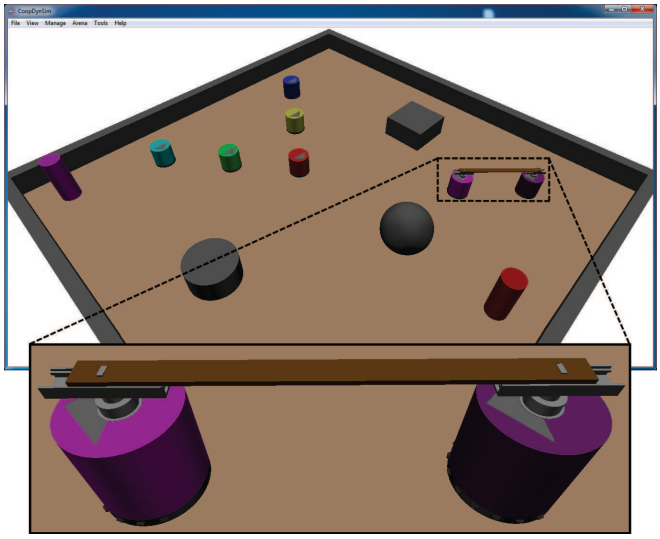

Fig. 4. Virtual environment with a team of 2 robots transporting a bar and a team of 5 robots in a formation.

The message's fields are as follows: *Error Code* is an integer that reports the success of the message; *Text* is a set of characters; *Command* is an integer that contains the command's id; *M Parameters* represents the number of parameters that the current message has, with *Parameter 1, 2, . . . , M* accounting for the parameters sent; in the same way, *N Data* and *Datum 1, 2, . . . , N* are the number of floating point values, and the values themselves, respectively. *M Parameters* and *N Data* are used to unpack the message on its destination. By following this message protocol, the user can control the robots using any language that supports socket based connections.

### A. Design

Concerning the main implementation scheme, *CoopDynSim* runs three main threads, one dedicated to the physics update, another dedicated to render the scene and the final one responsible for the interface window. Each of these threads runs independently from the others, with different update rates. For instance, the visualization thread does not require a high rate (we use 10 fps by default), whereas the physics one needs a greater update rate (around 100 fps in our case). Furthermore, each robot inserted in the simulator spawns a dedicated thread. Responsible for updating the values of the actuators (with the last command received) and sensors (with the last update from the physics), this thread can run independently from the physics thread, implying *real time* simulation, that is, the time elapsed in simulation is the same as the real time, and the simulation is independent from the control application(s) connecting to the robot(s). Conversely, if one wishes to speed up a simulation or increase the number of platforms in use, a *simulated time* option is available, where each iteration step from the part of the physics will only occur after each robot has received its control command (thus implying a synchronization mechanism), and the time to be simulated by the engine is a fixed value, independent from the elapsed time, which can be defined by the user (it can not be lower than 10 milliseconds, in order to guarantee physics stability).

As for the objects in the virtual world, each is composed of physical and graphical properties, Fig. 3. Physical properties are represented by the shape (simple shapes or composed ones), the mass, the mass distribution, friction coefficients, etc. All these properties are predefined, unless if the object is inserted using a configuration file (see Fig. 7), in which the user can specify its mass. Graphical properties are defined by the 3D shape to represent, i.e. the object's vertices and colors information. When an object is inserted in the virtual environment, the physics' thread "acts on it" and the graphics' thread updates its location. Hence, each object has a shared block of memory, accessed by both, that contains information about its location. The robotic platforms are nothing more than a set of attached objects, arranged in the best possible fashion, in order to accurately emulate the real robots.

The virtual robots have the same characteristics as the real ones, i.e. dimensions, sensors and actuators. Actually, two main types of robots were implemented at first. Composed of a cylindrical chassis, eleven distance sensors, two differential motorized wheels, two caster wheels, one type has, in addition, a dedicated support needed for cooperative transportation tasks. To emulate the distance sensors, a ray trace algorithm, provided by the physics engine, is used in each of the sectors. As for the locomotion, the motorized wheels are emulated by two cylinders attached to the main chassis by a hinge joint, with the caster wheels needed to balance the platform. The vision system is not being replicated, with the module simply returning the target (colored marker or another robot) information. Fig. 4 shows a team of two robots transporting a bar and its target (magenta cylinder), a team of five robots in a inverted V shape formation and its target (red cylinder), and 3 distinct obstacles (box, cylinder and sphere).

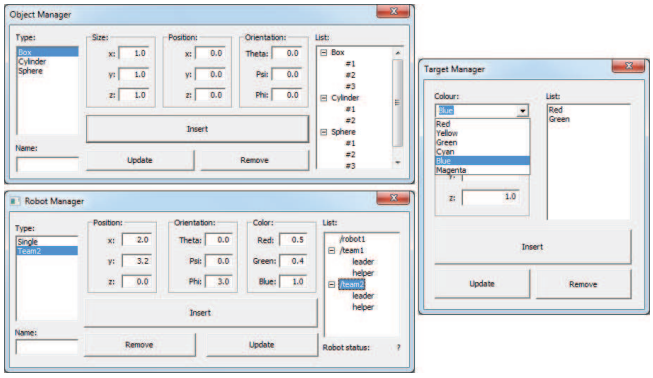


Fig. 5. Objects', Robots' and Targets' Managers.

Fig. 6. Virtual environment with two robotics arms: *ABB™IRB 120* (left side) and *Motoman™MH5* (right side) [21].

```
[FLOOR]
size        lenght width height
[WALL_1]
type        box_cylinder_or_sphere
size        lenght_wall_1 width_wall_1 height_wall_1
position    x_wall_1 y_wall_1 z_wall_1
orientation phi_wall_1 psi_wall_1 theta_wall_1
mass        m_wall_1
...
[WALL_n]
type        box_cylinder_or_sphere
size        lenght_wall_n width_wall_n height_wall_n
position    x_wall_n y_wall_n z_wall_n
orientation phi_wall_n psi_wall_n theta_wall_n
mass        m_wall_n
```

Fig. 7. World file template.

### B. User interaction

After describing the main components of the simulator and addressing its inner workings, the question of how the user can interact with the software arises. Starting with an empty world (with only the floor), navigation through the virtual environment is achieved via the mouse buttons: left button for rotation, right button for translation and mouse wheel for zooming in and out. To insert elements in the world, two levels of abstraction are available to the user.

*1) Basic elements:* Accessible through the *Manager* menu, these represent the three basic constituents of the virtual environment: *objects*, *robots* and *targets*.

*a) Objects:* With the *Object Manager* (Fig. 5 top left) the user can easily insert simple objects, such as a *Box*, a *Cylinder* and/or a *Sphere*, specifying the size, position and orientation within the virtual world. These can also be updated, at any time, or removed, through the same menu. Only three basic objects are available, nevertheless, the implementation of more complex ones is a trivial task, given the way the code is organized.

*b) Robots:* Single or teams of robots (the latter used for transportation tasks) can be inserted through the *Robot Manager* menu (Fig. 5 bottom left). In the same way, position and orientation can be modified, with the addition of a customizable color (useful to differentiate the robots from one another, in multi-robot scenarios) and name. This name identifies the virtual platform on the network, and each needs to have a different one. For instance, "/robot1/motors" and "/robot2/motors" represent the tags, on the YARP network, for the motors' module of robots 1 and 2, respectively.

Not only mobile platforms can be emulated though, with three robotic arms being added to the simulator at the time of writing (concretely, *amtec™lwa 7dof*, *ABB™IRB 120* and *Motoman™MH5*), which helps to prove the flexibility of the solution, when it comes to the type of platforms it can simulate (Fig. 6).

*c) Targets:* Representing a special type of objects, targets (*Target Manager*, Fig. 5, right) are colored landmarks which specify desired destinations for the robots to reach. Each mobile platform has a *target* module that returns the distance and angular displacement to these markers, thus (roughly) simulating the vision system (we use colored boxes, with the real platforms, to represent the desired location to reach in the real world).

*2) Compound elements:* In order to decrease the effort necessary to setup an experiment, more complex elements are available, concretely, *worlds* and *scenarios*.

*a) Worlds:* Composed of a floor and *N* objects, these *arenas* help the user to setup a custom environment, and easily load it into the simulator. Several default ones are already available via the *Arena* menu. In order to create a new arena, a plane text file with the *.world* extension is used. Fig. 7 shows the structure of such file, where the user can specify the *FLOOR* dimensions (*length*, *width* and *height*) and each of the object's properties (*type*, *size*, *position*, *orientation* and *mass*). These user defined arenas can be loaded using the *Load from file...* item in the *Arena* menu.

*b) Scenarios:* In order to quickly setup an experiment, in addition to a custom world, a complete scenario can also be loaded by the user (plane text file with *.scenario* extension file). The same file based approach is used here (its structure can be seen in Fig. 8), where the world, targets and robots (type and properties) can also be defined. This option can be accessed through the *Load Scenario...* item in the *File* menu.

### C. Play back mode

A useful feature implemented in the simulator concerns the play back option. If the user chooses to, in each iteration step, the software will save the robots' positions within the virtual world to a specific file, as well as the scenario used.

```
[WORLD]
file          path_to_the_world_file
[TARGET]
color_1       x_pos_tar_1 y_pos_tar_1 z_pos_tar_1
...
color_n       x_pos_tar_n y_pos_tar_n z_pos_tar_n
[ROBOT_1]
type          single_or_team2
name          network_name_of_the_robot_1
position      x_robot_1 y_robot_1 z_robot_1
orientation   phi_robot_1 psi_robot_1 theta_robot_1
color         R_1 G_1 B_1
...
[ROBOT_N]
type          single_or_team2
name          network_name_of_the_robot_n
position      x_robot_n y_robot_n z_robot_n
orientation   phi_robot_n psi_robot_n theta_robot_n
color         R_n G_n B_n
```

Fig. 8. Scenario file template.

Afterwards, to recap the simulation, a load menu is prompted to the user, and the positions from the play back file are stored in memory. In order to go to a specific time in the past simulation, a global module based on YARP is used, which receives said desired time via the network (much like the modules of the robotic platforms).

Analyzing the dynamics of the robots (along with log files from the control application), whilst having a visual feedback of the positions of the robots in the world, makes the process of debugging a control approach a much easier task, especially in a multi-robot scenario.

## IV. Use cases

*CoopDynSim* started being developed to be used with object transportation tasks by multi-robot teams [22]. Fig. 9 illustrates a simulation with two teams of two robots transporting a long bar from an arena's corner to the opposite one. Here, each team features a *leader*, whose main responsibility is reaching the final destination, and a *helper*, who helps the leader carry the load. By making use of the simulated support (custom made sensor) and replicating a complex joint transportation task, this scenario is particularly important when it comes to endorse the flexibility of the simulator.

The software was also used in a multi-robot formation research [11], in which teams of autonomous mobile robots navigate in a desired configuration, whilst avoiding obstacles that may lie in the robots' path, by breaking formation, returning to their position after such obstacles are surpassed (Fig. 10 depicts such scenario).

A different project in development, using the software, is aimed at using robotic arms to aid in brain surgery. By adding different arms to the simulator, the study of which is more appropriate to the task in hand becomes easier, as well as the testing of different control algorithms, for such a delicate application.

Concerning its applicability as a teaching tool, it was used for the first time in the *Hands-on Summer School: Neural Dynamics Approaches to Cognitive Robotics 2011* [10]. The participants quickly became familiar with the software, and a few transferred the code that themselves wrote (in the control application) for the simulation, to the real platforms, with only a few parameters' adjustment needed to obtain the same results as in simulation. This simplicity and complete transparency of the whole approach made the simulator receive a great feedback from the participants.

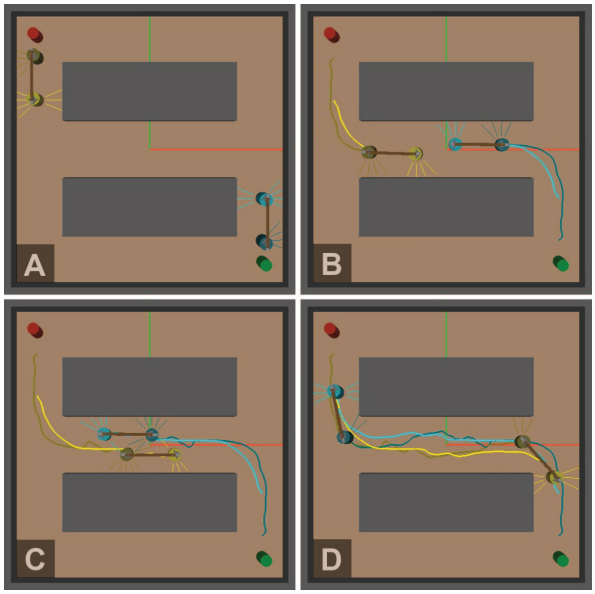


Fig. 9. Snapshots taken from a simulation of a team of two robots transporting long loads [23].

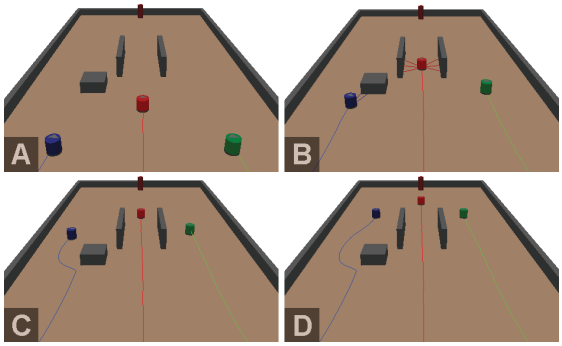


Fig. 10. Snapshots taken from a simulation of a team of three robots navigating in a triangle formation [11].

After such successful use of *CoopDynSim*, it was adopted as a teaching tool for the *Automation, Control and Robotics* in the Industrial Electronics Engineering Master Degree, at the University of Minho in Portugal. Here, the students are given a single install package for the simulator and a MATLAB application to implement the control. The software helps the students to better understand the theoretical concepts taught in the course.

## V. Conclusions

We have presented *CoopDynSim*, a 3D robotics simulator, based on Newton Game Dynamics, Open Graphics Library and YARP.

By having complete transparency, when it comes to the controller code, as one of the design criteria, the simulator greatly diminishes both the effort and the time spent migrating from simulation to the real implementation. With this dedicated solution, but easily expandable to include other robot models, when the control application is transferred to our real platforms just fine tunning control parameters may be needed (real robots are accurately replicated in the simulator, but obviously some unexpected real perturbations can not be taken into account). If robot models provided by the most other simulators were used, the switch to real platforms would exhibit an exaggerated effort which increases with the number of robots (multi-robot control) due to the sensory information and motor actuation are distinct from the one of the real platforms.

Furthermore, the possibility to easily add custom made sensors and platforms (albeit having to directly modify the source code), succors the flexibility of the solution, widening the possible applications in various research projects.

The *simulated time* and *real time* options further increase the software's flexibility, since the former can be used for a simulation with a high number of platforms (for instance, swarms) and the latter is more suitable to use in scenarios with only a few robots. Also, the *play back* feature gives the user a possibility to recall an entire simulation, which represents an useful feature of the software.

The simulator's user friendliness and simplicity were validated by its usage as a teaching tool in our University.

*CoopDynSim* is free and can be downloaded from our MARL web server [1].

## VI. Future work

*CoopDynSim* is still in an early development stage. The main requirements for the project have been fulfilled, nevertheless, many aspects can be improved and some features can be added.

In order to make the simulator more visually appealing, some textures could be added to the objects on the world. Furthermore, the possibility to easily add new objects and/or robots without making changes to the source code, as well as a, more intuitive, drag and drop user interface are key features found in many available simulators that are lacking in our solution.

The simulated vision system is another point that needs improvement. Instead of just directly returning the angle and distance to a target, a virtual image of the robot's field of vision (virtual camera) will give the possibility to use the same image processing application that is in use in the real robots.

Moreover, the software should be made platform independent (i.e. Linux, Mac OS, etc), liberating the user from having to use a specific system (currently it only runs on Windows OS).

## Acknowledgments

We would like to thank all the people that work in our laboratory, MARL (Mobile and Anthropomorphic Robotics Laboratory) in University of Minho. All of them contributed in several ways for the success of this work. A special thanks to Carlos Faria, for the addition of the three robotic arms to the simulator. Toni Machado would also like to thank the Portuguese Science and Technology Foundation for providing his Ph.D. scholarship (ref. SFRH/BD/38885/2007).

[1] http://marl.dei.uminho.pt/Public/CoopDynSim.zip

## References

[1] "Workshop on robot simulators: available software, scientific applications and future (along with IROS)," Nice, France, 2008.
[2] "Workshop on Space Robotics Simulation (along with IROS)," 2011.
[3] "International Conference on Simulation, Modeling and Programming," Darmstadt, Germany, 2010.
[4] "International Conference on Modelling and Simulation," Phuket Island, Thailand, 2011.
[5] J. Craighead, R. Murphy, J. Burke, and B. Goldiez, "A Survey of Commercial & Open Source Unmanned Vehicle Simulators," in *2007 IEEE International Conference on Robotics and Automation*, no. April, 2007, pp. 852–857.
[6] P. Castillo-Pizarro, T. V. Arredondo, and M. Torres-Torriti, "Introductory Survey to Open-Source Mobile Robot Simulation Software," in *2010 Latin American Robotics Symposium and Intelligent Robotics Meeting*. Ieee, Oct. 2010, pp. 150–155.
[7] A. Harris and J. M. Conrad, "Survey of Popular Robotics Simulators , Frameworks , and Toolkits," in *2011 Proceedings of IEEE Southastcon*, 2011, pp. 243–249.
[8] "Newton Game Dynamics." [Online]. Available: http://newtondynamics.com
[9] "Open Graphics Library." [Online]. Available: http://www.opengl.org
[10] "Summer School Neuronal Dynamics Approaches to Cognitive Robotics," 2011. [Online]. Available: http://cognitive-robotics-school.dei.uminho.pt
[11] M. Sousa, "Multi-robot cognitive formations," M.Sc. dissertation, University of Minho, 2011.
[12] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems," in *International Conference on Advanced Robotics ( ICAR 2003 )*, no. Icar, 2003, pp. 317–323.
[13] "Open Dynamics Engine." [Online]. Available: http://www.ode.org
[14] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "USARSim: a robot simulator for research and education," in *2007 IEEE International Conference on Robotics and Automation*, no. April, 2007, pp. 1400–1405.
[15] L. Hugues and N. Bredeche, "Simbad: An Autonomous Robot Simulation Package for Education and Research," *Lecture Notes in Computer Science*, vol. 4095, pp. 831–842, 2006.
[16] T. Laue, K. Spiess, and R. Thomas, "SimRobot - A General Physical Robot Simulator and Its Application in RoboCup," *Lecture Notes in Computer Science*, vol. 4020, pp. 173–183, 2006.
[17] J. Jackson, "Microsoft Robotics Studio: A Technical Introduction," *IEEE Robotics & Automation Magazine*, no. December, pp. 82–87, 2007.
[18] O. Michel, "Webots: Professional Mobile Robot Simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, pp. 39–42, 2004.
[19] "Cogmation Robotics: robotSim documentation." [Online]. Available: http://www.cogmation.com/pdf/robotsim_doc.pdf
[20] P. Fitzpatrick, G. Metta, and L. Natale, "Towards long-lived robot genes," *Robotics and Autonomous Systems*, vol. 56, no. 1, pp. 29–45, 2008.
[21] C. Faria, "Robotic implantation of intracerebral electrodes for deep brain stimulation," M.Sc. dissertation under development, University of Minho, 2012.
[22] R. Soares, E. Bicho, T. Machado, and W. Erlhagen, "Object transportation by multiple mobile robots controlled by attractor dynamics: theory and implementation," in *Proceedings on the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*. San Diego, CA, USA: IEEE, Oct. 2007, pp. 937–944.
[23] T. Machado, T. Malheiro, S. Monteiro, and E. Bicho, "A dynamical systems approach to flexible transportation by teams of two robots," *Manuscript under preparation*, 2012.