DOI: http://dx.doi.org/10.7551/978-0-262-31709-2-ch078

FARSA[1]: An Open Software Tool for Embodied Cognitive Science

Gianluca Massera[2], Tomassino Ferrauto[2], Onofrio Gigliotta[3], Stefano Nolfi[2]

[1]http://laral.istc.cnr.it/farsa
[2]Institute of Cognitive Sciences and Technologies, CNR, Rome, Italy
[3]University of Naples Federico II, Naples, Italy
{gmassera, tomassino.ferrauto, s.nolfi}@istc.cnr.it, onofrio.gigliotta@unina.it

Abstract

In this paper we introduce the FARSA open-source tool that supports the accomplishment of experimental research in Embodied Cognitive Science and Adaptive Behavior. The tool provides a set of integrated libraries and a graphical interface that enable to design, to accurately simulate and to analyze individual and/or collective embodied robotic models. The modular architecture of the tool allows to progressively expand it with new software components and simplifies the implementation of custom experiments. The tool comes with a set of exemplificative experiments and with a synthetic but comprehensive documentation that should enable users to quickly master its usage.

Introduction

The realization of the importance of embodiment and situatedness for the study of behavior and cognition led to a paradigm shift toward the so-called Embodied Cognitive Science (Clark, 1999; Pfeifer and Bongard, 2007; Shapiro, 2007). From a methodological point of view this change implies that models of behavioral and cognitive capacities should take into consideration the characteristics of the agent's nervous system, of the agent's body, of the environment and of the properties that arise from the agents/environment interactions. This in turn requires the formulation of models that are far more complex than their previous disembodied counterpart and that are not constituted simply by static descriptions but rather by processes that run in the physical world or in realistic computer simulations.

The recent development of robotic platforms that are relatively affordable and easy to use (such as the Khepera[1] and the Nao[2] robot) as well as the development of software libraries that enable the realization of realistic simulations of physical processes (such as ODE (Smith, 2004) and Newton Dynamics (Jerez and Suero, 2004)) constitute important facilitators for the design of embodied models. Despite of that, the knowledge barrier that Embodied Cognitive Science researchers should face to build and analyze their models is still very high.

To mitigate this problem we developed FARSA, an open-source software tool that enables researchers and students to easily and effectively carry on research in Embodied Cognitive Science. FARSA combines in a single framework the following features:

- it is open-source, so it can be freely modified, used and extended by the research community;
- it is constituted by a series of integrated libraries that allow to easily design the different components of an embodied model (i.e. the agents' body and sensory-motor system, the agents' control systems, and the ecological niche in which the agents operate) and that allow to simulate accurately and efficiently the interactions between the agent and the environment;
- it comes with a rich graphical interface that facilitates the visualization and analysis of the elements forming the embodied model and of the behavioral and cognitive processes originating from the agent/environment interactions;
- it is based on a highly modular software architecture that enables a progressive expansion of the tool features and simplifies the implementation of new experiments and of new software components;
- it is multi-platform, i.e. it can be compiled and used on Linux, Windows, and Mac OS X operating systems;
- it comes with a set of exemplificative experiments and with a synthetic but comprehensive documentation that should enable users to quickly master the tool usage.

In section 2 we discuss the relation with other similar tools. In section 3 we review the main features and capabilities of the tool. In section 4 we describe the design and working principles of the software architecture. Finally in section 5, we describe the planned future extensions of the tool.

[1]http://www.k-team.com/mobile-robotics-products/khepera-ii
[2]http://www.aldebaran-robotics.com/en/

Related Tools

Objectives similar to those we have tried to reach with FARSA have been actively pursued during the last 20 years by academic research laboratories, small private companies, and multinational companies. Here we restrict our analysis to the most related attempts.

One of the first and most influential tool is Webots™ (Michel, 2004) a mobile robot simulator, initially developed by Olivier Michel at the Swiss Federal Institute of Technology (EPFL) in Lausanne, Switzerland, and then commercialized by a small spin-off company led by the software creator. The tool is used by about 1000 research centers and universities worldwide. Webots™includes a robot simulator with a rich collection of predefined robotic models, a visualization tool that allows to observe the robot's behavior, a library of methods for customizing the robot and the environment, and a library of simulated sensors and actuators. A limitation of this tool is constituted by its commercial nature that introduces a cost barrier, prevents the possibility to fully inspect and customize the source code, and limits the possibilities to exploit collaborative development.

ARGOS (Pinciroli et al., 2012) is a recently developed 3D physic simulation tool targeted particularly toward swarm robotics research. It is an open source project. The usage of the tool, however, require a significant programming effort also due to the lack of an integrated graphical interface.

USARSim (Carpin et al., 2007) is also open-source simulator that was initially targeted toward urban search and rescue scenarios and later extended toward a more general use. It supports a wide range of robotics platforms (humanoids, wheeled, vehicles, etc) and has been adopted as a simulation platform by the Robotcup initiative. USARSim is an open source project which, however, is based on the Unreal Engine proprietary technology. This limits the inspection and the customization of the tool at the level of the robot implementation and impose the use of a particular proprietary language (unreal script) for the configuration of the experiments.

Gazebo (Koenig and Howard, 2004) is another open-source simulator anlogous to USARsim but does not use any third party proprietary code. Another advantage of Gazebo is that it can be used in combination with Player (Gerkey et al., 2001) a tool that can be used to design the agents' controller and eventually the adapting process. The two tools are constituted by indepedent software programs that communicate through a dedicated network protocol. The combination of these tools constitute a powerful environment. Its use, however, requires a significant programming expertise and learning efforts.

Finally, Microsoft and Willow Garage developed two similar robotic suites: Microsoft Robotics Developer Studio (RDS)[3] and Robot Operating System (ROS)[4]. These packages, that constitute a sort of operating system and a developmental environment for robotics applications, include device drivers, libraries, visualizers, message-passing, package management, 3D simulation tools, visual programming languages, a library of simulated robotic platforms, sensors, and actuators. The limits of these tools, for what concerns the objective addressed in this paper, is their complexity and consequently the required expertise and learning efforts. Another technical limitation is constitued by the fact that they are not multi-platform: Microsoft RDS runs only on windows while ROS runs only on Linux, limiting community collaborations.

The FARSA project aims to provide an open-source and multi-platform tool that it is easy to use and to extend and, in addition to a robotic simulation environment, provides integrated tools for designing the control systems of the robots, for analysing the robots' behaviour, and for subjecting the robots to evolutionary and/or learning processes.

Features and Capabilities

FARSA is a re-engineered and extended version of a tool that has been developed since the 1995 by Stefano Nolfi and Onofrio Gigliotta (Nolfi, 2000; Nolfi and Gigliotta, 2010) which has been used for research and education purposes by more than 50 research laboratories and universities. It is in an open-source software tool that can be freely used and modified and a cross-platform application that runs on Linux, Windows and Mac OS X (on either 32bit or 64bit systems). The tool can be downloaded from http://laral.istc.cnr.it/farsa. FARSA is well documented, easy to use and comes with a series of exemplificative experiments that allow users to quickly gain a comprehension of the tool. These experiments can be used as a base for running a large spectrum of new experiments that can be set up simply by editing a configuration file.

The tool is constituted by a series of integrated software libraries providing the features described in the following sub-sections.

The Robots/Environment Simulator

The robots/environment simulator (*worldsim*) is a library that allows to simulate the robot/s and the environment in which it/they operate. The library supports both individual robot simulation and collective experiments in which several robots are placed in the same environment. The physical and dynamical aspects of the robots and of the robots/environment interactions can be simulated accurately by using a 3D dynamics physics simulator or by using a faster but simplified kinematic engine. For what concern the dynamics simulation, FARSA relies on the Newton Game

[3] http://www.microsoft.com/robotics/

[4] http://www.willowgarage.com/pages/software/ros-platform

Dynamics engine (Jerez and Suero, 2004) that enables accurate and fast simulations. The underlying dynamic engine has been encapsulated so to enable the inclusion of alternative engines.

Currently, FARSA supports the following robotic platforms: the Khepera (Mondada et al., 1994), the e-Puck (Mondada et al., 2009), the marXbot (Bonani et al., 2010) (see Figure 1, bottom) and the iCub (Sandini et al., 2004) (see Figure 1, top). These robots have been designed by assembling a series of building blocks (physical elements, sensors, and motorized joints) that users can re-use to implement alternative, not yet supported, robots.

In the case of the iCub, the simulator is based on the YARP (Metta et al., 2006) middleware library (the same command used to read the robot's sensors and control the robot's motor can be used to work with the simulated or real robot). This strongly facilitates the possibility to port results from simulation to reality and the possibility to integrate into FARSA projects the software modules available from the iCub software repository[5]. With respect to the iCub simulator developed by Tikhanoff et al. (2008), the simulation library included in FARSA presents a series of advantages: it strictly conforms to the real kinematic joints structure of the robot, it allow to simulate multiple robots, it includes both a dynamic and kinematic engine, and it provides an enhanced visualization tool.

The Sensor and Motor Library

FARSA also includes a library of ready-to-use sensors and motors. In some cases, sensors and motors include software routines that pre-elaborate sensory or motor information (e.g. to reduce its dimensionality) and/or integrate different kinds of sensory-motor information (as in the case of motors that set the torque to be produced by a joint motor on the basis of the current and desired position of the controlled joint).

Wheeled robots are provided with infrared, ground, traction force, linear vision, and communication sensors, among others. Moreover, they are provided with wheels, grippers, LEDs, and communication actuators.

The iCub robot is provided with proprioceptors that measure the current angular position of the robot's joints, tactile sensors, and vision sensors among others and with actuators that control all the available DOFs.

The state of the robot's sensors and motors, as well as the state of selected variables of the robot's control system, can be graphically visualized while the robot interacts with the environment (see Figure 2). This provides an useful analysis and debugging tool.

The Controller Libraries

These libraries enable the user to design, modify and visualize the robot's control system. Currently FARSA includes

[5]http://wiki.icub.org/iCub_documentation/

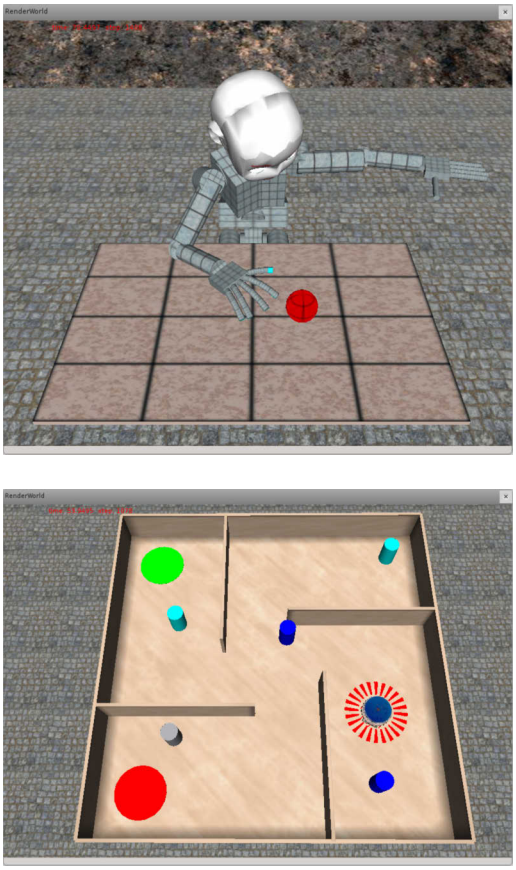

Figure 1: Snapshots taken from the 3D robot/environment renderer of FARSA. Top: A simulated iCub robot that reaches and grasps a spherical object located over a table. Bottom: A simulated marXbot robot that navigates in a structured environment containing walls and coloured objects.

two libraries that support the design of neuro-controllers. Users willing to use other architectures or formalisms can integrate into FARSA alternative libraries (see the section "Extending FARSA").

Evonet is an easy-to-use library that enables users to graphically design, modify and visualize the architecture of the robot's neural controller as well as the properties of the neurons and of the connection weights (see Figure 2). The library supports logistic, leaky integrator, and threshold neurons. NNFW is an alternative object-oriented library that provides a larger variety of neuron types and output functions (Gaussian, winner-take-all, ramp, periodic, etc.) and supports the use of radial basis function neural network.

Thanks to the integration between the controller and the sensory and motor libraries, the sensory and motor layer of the neural controller is automatically generated on the basis of the selected sensors and motors. Moreover, the update of the sensory neurons and the update of the actuators on the basis of the state of the motor neurons is handled automatically.

Finally, the graphic viewer of the robot's controller (see Figure 2) also enables users to lesion and/or to manually manipulate the state of the sensors, internal, and motor neurons in order to analyze the relationship between the state of the controller and the behavior that originates from the robot/environmental interaction.

The Adaptation Libraries

These libraries enable the user to subject a robot or a population of robots to an adapting process (i.e. to a evolutionary and/or learning process during which the characteristics of the robots are varied and variations are selected so to improve the abilities of the robots to cope with a given task/environment).

The adaptation libraries that are currently available support the use of evolutionary algorithms (including steady state, truncation selection, and Pareto-front algorithms), supervised learning algorithms (i.e. back-propagation), and unsupervised learning algorithm (i.e. Hebbian learning). The evolutionary algorithms are parallelized at the level of the individual's evaluation and can therefore run significantly faster in multi-core machines and computer clusters.

In the case of evolutionary and supervised algorithm, the variation in performance during the adaptation can be monitored and analyzed in the associated graphic renderer (see Figure 3).

Design and Working Principles

The architecture of FARSA is based on three key ideas: the *components*, the *configuration file* and the *plugins*.

The *components* are software modules that implement a given object or process. They can be organized in a hierarchical manner. For example, a project might include an

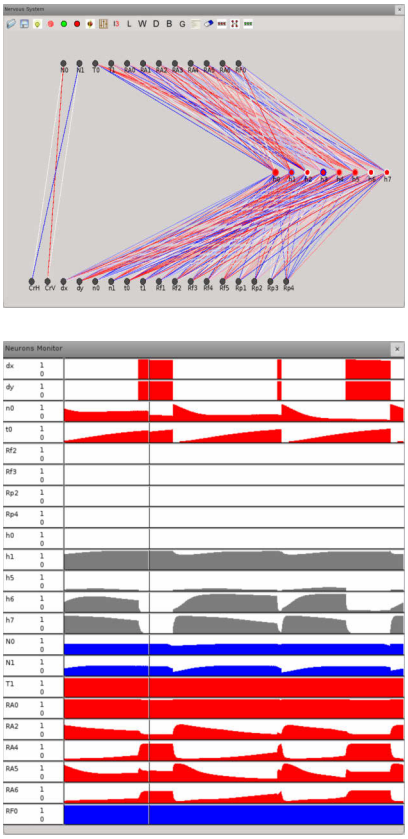

Figure 2: Top: The controller graphic widget that allows to visualize, modify, and analize the robot's neural architecture, the strenght of the connection weights and biases, and the properties of the neurons. Bottom: The controller monitor that displays the activation state of the sensory, internal, and motor neurons while the robot interacts with the environment.

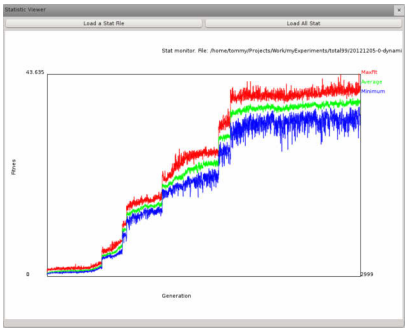

Figure 3: The graphic widget of the adapting process. In this example, the widget is used to show the best, average and worst fitness of an evolutionary experiment through out generations.

evolutionary process component, that includes as subcomponent an experimental component, that includes as subcomponent an iCub robot component, a neural network controller component, and several sensors and motors components. The main characteristic of components is that they can be automatically instantiated and configured from the content of a *configuration file* (they have a direct relation to groups of parameters in a *configuration file*, as explained below). Components might also include associated commands (e.g. <"evolve", "stop", "test"> in the case of an evolutionary component), and graphical widgets that can be accessed by the FARSA main graphic interface (see next section).

The *configuration file* is a text file that specifies the components (e.g. the robotic platform, the robots' sensors and the motors, the robots' controllers, and eventually the robots' adapting process) that are going to be used in a particular project and the parameters (e.g. the number of robots situated in the same environment, the number and type of objects present in the environment) that are used to configure them. The file has a hierarchical structure analogous to the hierarchial organization of components. The configuration file is a human readable text file (in .INI or .XML format) that can be edited through the Total99 graphic interface (described in the next section) or directly through a standard text editor. This enables users to configure and run experiments on remote machine (e.g. computer clusters) that do not have a graphical environment. The modular and hierarchical organization of components combined with the configuration file has several advantages:

- it allows to instantiate a runtime only the components that are needed in a particular project, thus eliminating the risk that problems affecting other components might affect the functionality of the whole project,
- it gives the possibility to re-use the same components in different projects,
- it enables a progressive expansion of the tool with the development of additional components,
- it simplifies the tool usage through the visualization of only the parameters, the commands, and the graphic widgets that are relevant for a given project.

A *plugin* contains compiled code of new components or features created by users. They might consists of subclasses of existing components (e.g. a subclass of an evolutionary experiment with a new implemented fitness function or a new subclass of the sensor class implementing a new type of sensor not available in the sensor library) or of completely new components (e.g. a behaviour-based controller tool with associated parameters, commands and graphic widgets). The plugins, which are loaded and instantiated at run time, are totally equivalent to the other native components of FARSA for what concern the functionalities and use (e.g. they can be configured and commanded in the same manner and through the same graphic interface of the native components). Plugins provide several advantages:

- they enable users to neatly separate their new code from the main library,
- they facilitate the distribution and sharing of additional components and feature within the FARSA community,
- they enable users to get access to a number of exemplificative experiments that increase over time,
- they allow authors of scientific papers to provide an easy way to replicate their work.

Overall the workflow in FARSA is as follow: the project configuration file and the required plugins are loaded, the required components are created and configured on the basis of the configuration parameters, the associated commands and graphical widgets are created and made available to the user through the graphic interface.

The graphical interface

Total99 is the graphical interface that allows to configure experiments, to instantiate the required software components, and to use the associated commands and graphic widgets. Total99 can also operate in batch mode without graphics if required. Total99 can be used to create, view, or modify a configuration file (Figure 4). This can be done by loading or creating a configuration file (through the use of the commands available in the File menu) and by then setting the configuration components and parameters through the parameters widget (orange rectangle of Figure 4).

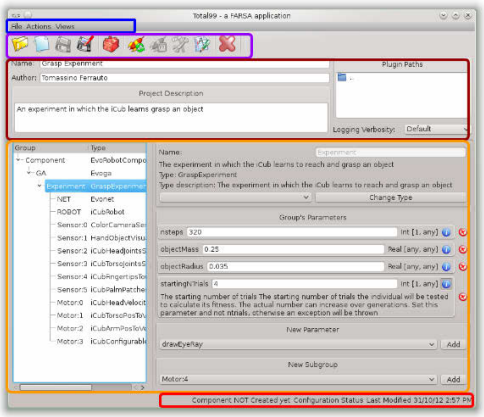

Figure 4: The Total99 graphical interface. The menu bar (blue), the toobar (magenta), the project information bar (brown), the project parameters widget (orange), and the status bar (red) have been highlighted with coloured rectangles.

More specifically, the left part of the parameters widget is used to display the hierarchical organization of the components and the right part is used to display the parameters of the currently selected component and/or to add or remove sub-components and parameters (these can be selected from automatically generated lists that include only the parameters that belong to the current component and the sub-components that can be instantiated from the current component).

Once the configuration file has been set up, the user can run the project through the menu or the tool bar. As we mentioned above, this initiates the loading of the selected plugins, the instantiation of the software components specified in the configuration file, and the configuration of the components on the basis of the parameters specified in the configuration file. At this point, the commands associated to the components that have been instantiated and the associated graphic widgets can be executed from the Action and Views folders of the menu bar.

Extending FARSA

FARSA can be extended by implementing additional software components. The integration of the new components into FARSA, that enables the possibility to use, execute, and configure them as native components, requires to fulfill the following three requirements.

The first requirement is that the class of the new component should be defined as a sub-class of an existing component or of a virtual empty component.

The second requirement is that the new class should include a describe(), save(), and configure()[6] functions that are used respectively for declaring the properties of the configuration parameters, saving the current value of the parameters on the configuration file when requested, and configuring the object on the basis of the parameters. For all these operations, FARSA provides helper functions in order to simplify and to minimize the effort of writing them.

The third requirement is that the new component has to be registered with the registerClass() function specifying the component type name.

In the case of components that also include new commands and/or new graphic widgets (that should be made available from the Total99 graphic interface), the user should also declare the new commands and/or widgets by implementing the fillActionsMenus(), getViewers(), and addAdditionalMenus() functions.

More detailed information can be found in the on-line documentation.

Future Plans

The first stable version of FARSA has been just released online. During the next few months we plan to refine and extend the documentation and the library of the exemplificative experiments. Then we plan to keep extending the tool, to promote the development of a community of users and developers, and to develop multimedia materials that can enrich the educational and training potential of the tool.

Planned Extensions

Currently planned extensions include: (i) additional controller and adaptive software components that will enable the direct use of other formalisms and training algorithms (e.g. self-organizing maps, fuzzy networks, reinforcement learning) as well as the combined used of different techniques (e.g. unsupervised and supervised learning algorithms), (ii) additional readily available robotic platforms, and (iii) the possibility of using a simple script language as an alternative to C++ to implement new types of experiments.

FARSA Community

The full open-source nature of FARSA and its modular architecture constitute two important prerequisites for the development of a wide community of users and developers that can profit from the tools and can contribute to its further extension. To enable the formation of such community we plan to widely disseminate the tool and to create web-based facilities that can be used to store users' knowledge and plugins.

Educational Use of FARSA

The simplicity of use potentially enables FARSA to also become an excellent educational tool. To promote this use we plan to develop and promote the collaborative development of tutorials and training material targeted toward undergraduate and graduate courses in Embodied Cognitive Sciences and Autonomous Robotics. Finally we plan to develop examples of serious games (Marsh, 2011; Miglino et al., 2008, 2007) that could be used to disseminate key concepts also to the general public, within Science Museums and Festivals, and to students of the primary and secondary schools.

Conclusion

FARSA is an easy to use open-source tool that can enable students and researchers with a limited technical expertise to start building and experimenting with embodied cognitive science models and enables experienced researchers to use a powerful tool that can be easily configured and extended. Moreover, it provides an unique set of integrated tools that strongly facilitate the design of neuro-robotics and adaptive robots. The current version of the tool has been extensively used and tested to carry on frontier research in adaptive robotics (Massera et al., 2007; Gigliotta and Nolfi, 2007; Tuci et al., 2009; Massera et al., 2010; Tuci et al., 2011; Savastano and Nolfi, 2012; Leugger and Nolfi, 2012) in our lab. We hope the public and well-documented version of the tool that we just released will attract a wide interest

[6] Instead of implementing the configure() function, it is possible to implement a special constructor of the class

and will permit the establishment of a wide community of users and developers.

Acknowledgments

This research has been supported by CNR, in part under the European Science Foundation project Hierarchical Heterogeneous SWARM (H2Swarm). The authors thank Piero Savastano, Fabrizio Papi, and Tobias Leugger who contributed to the development of the tool.

References

Bonani, M., Longchamp, V., Magnenat, S., Re?tornaz, P., Burnier, D., Roulet, G., Vaussard, F., Bleuler, H., and Mondada, F. (2010). The marXbot, a miniature mobile robot opening new perspectives for the collective-robotic research. *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4187–4193.

Carpin, S., Lewis, M., Wang, J., Balakirsky, S., and Scrapper, C. (2007). USARSim: a robot simulator for research and education. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1400–1405. Ieee.

Clark, A. (1999). An embodied cognitive science? *Trends in cognitive sciences*, 3(9):345–351.

Gerkey, B. P., Vaughan, R. T., Stø y, K., Howard, A., Sukhatme, G. S., and Mataric, M. J. (2001). Most valuable player: A robot device server for distributed control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001)*, number Iros, pages 1226–1231.

Gigliotta, O. and Nolfi, S. (2007). Formation of spatial representations in evolving autonomous robots. *2007 IEEE Symposium on Artificial Life*, pages 171–178.

Jerez, J. and Suero, A. (2004). Newton Game Dynamics. http://www.newtondynamics.com.

Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2149–2154. Ieee.

Leugger, T. and Nolfi, S. (2012). Action development and integration in an humanoid icub robot. In *Biomimetic and Biohybrid Systems*, pages 369–370. Springer.

Marsh, T. (2011). Serious games continuum: Between games for purpose and experiential environments for purpose. *Entertainment Computing*, 2(2):61 – 68.

Massera, G., Cangelosi, A., and Nolfi, S. (2007). Evolution of prehension ability in an anthropomorphic neurorobotic arm. *Frontiers in neurorobotics*, 1:1–9.

Massera, G., Tuci, E., Ferrauto, T., and Nolfi, S. (2010). The Facilitatory Role of Linguistic Instructions on Developing Manipulation Skills. *IEEE Computational Intelligence Magazine*, 5(3):33–42.

Metta, G., Fitzpatrick, P., and Natale, L. (2006). Yarp: yet another robot platform. *International Journal on Advanced Robotics Systems*, 3(1):43–48.

Michel, O. (2004). Webots: professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):39–42.

Miglino, O., Gigliotta, O., Cardaci, M., and Ponticorvo, M. (2007). Artificial organisms as tools for the development of psychological theory: Tolman's lesson. *Cognitive Processing*, 8:261–277.

Miglino, O., Gigliotta, O., Ponticorvo, M., and Stefano, N. (2008). Breedbot: an evolutionary robotics application in digital content. *The Electronic Library*, 26(3):363–373.

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J.-C., Floreano, D., and Martinoli, A. (2009). The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th conference on autonomous robot systems and competitions*, volume 1, pages 59–65.

Mondada, F., Franzi, E., and Ienne, P. (1994). Mobile robot miniaturisation: A tool for investigation in control algorithms. In *Experimental Robotics III*, pages 501–513. Springer.

Nolfi, S. (2000). Evorobot 1.1 User Manual. Technical report, Institute of Psychology, CNR.

Nolfi, S. and Gigliotta, O. (2010). Evolution of Communication and Language in Embodied Agents. pages 297–301.

Pfeifer, R. and Bongard, J. (2007). *How the body shapes the way we think: a new view of intelligence*. Bradford Books.

Pinciroli, C., Trianni, V., O?Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Caro, G., Ducatelle, F., Birattari, M., Gambardella, L. M., and Dorigo, M. (2012). ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intelligence*, 6(4):271–295.

Sandini, G., Metta, G., and Vernon, D. (2004). Robotcub: An open framework for research in embodied cognition. In *Proceedings of IEEE/RAS International Conference on Humanoid Robots*, pages 13–32.

Savastano, P. and Nolfi, S. (2012). Incremental learning in a 14 dof simulated icub robot: Modeling infant reach/grasp development. In *Biomimetic and Biohybrid Systems*, pages 250–261. Springer.

Shapiro, L. (2007). The Embodied Cognition Research Programme. *Philosophy Compass*, 2(2):338–346.

Smith, R. (2004). Open Dynamics Engine. http://www.ode.org.

Tikhanoff, V., Cangelosi, A., and Fitzpatrick, P. (2008). An open-source simulator for cognitive robotics research: the prototype of the iCub humanoid robot simulator. In *Proceedings of IEEE Workshop on Performance Metrics for Intelligent Systems Workshop (PerMIS'08)*.

Tuci, E., Ferrauto, T., Zeschel, A., Massera, G., and Nolfi, S. (2011). An Experiment on Behavior Generalization and the Emergence of Linguistic Compositionality in Evolving Robots.

Tuci, E., Massera, G., and Nolfi, S. (2009). Active categorical perception in an evolved anthropomorphic robotic arm.