Simulating complex robotic scenarios with MORSE

Gilberto Echeverria[1⋆], Séverin Lemaignan[1,2⋆⋆], Arnaud Degroote[1⋆⋆⋆], Simon Lacroix[1†], Michael Karg[2‡], Pierrick Koch[3§], Charles Lesire[4¶], and Serge Stinckwich[3,5‖]

[1] CNRS, LAAS, 7 avenue du colonel Roche, F-31077 Toulouse, France / Université de Toulouse, UPS, INSA, INP, ISAE, LAAS, F-31077 Toulouse, France
[2] Institute for Advanced Studies, Technische Universität München, D-85748 Garching, Germany
[3] UMR 6072 GREYC Université de Caen-Basse Normandie/CNRS/ENSICAEN, France
[4] ONERA – the French Aerospace Lab, F-31055, Toulouse, France
[5] UMI 209 UMMISCO
IRD/IFI/Vietnam National University, Vietnam

Abstract. MORSE is a robotic simulation software developed by roboticists from several research laboratories. It is a framework to evaluate robotic algorithms and their integration in complex environments, modeled with the Blender 3D real-time engine which brings realistic rendering and physics simulation. The simulations can be specified at various levels of abstraction. This enables researchers to focus on their field of interest, that can range from processing low-level sensor data to the integration of a complete team of robots. After nearly three years of development, MORSE is a mature tool with a large collection of components, that provides many innovative features: software-in-the-loop connectivity, multiple middleware support, configurable components, varying levels of simulation abstraction, distributed implementation for large scale multi-robot simulations and a human avatar that can interact with robots in virtual environments. This paper presents the current state of MORSE, highlighting its unique features in use cases.

1 Introduction

Simulations are an essential component in robotics research, allowing the evaluation and validation of many sorts of developments before their integration

⋆ gechever@laas.fr
⋆⋆ slemaign@laas.fr
⋆⋆⋆ adegroot@laas.fr
† slacroix@laas.fr
‡ kargm@in.tum.de
§ pierrick.koch@unicaen.fr
¶ charles.lesire@onera.fr
‖ serge.stinckwich@ird.fr

on-board real robots. For such validations to be useful, the simulation must provide enough fidelity with respect to the real world, within the requirements of the considered robotics application. Creating a fully realistic simulation in every aspect remains nearly impossible and sometimes can also be seen as an inconvenience. Therefore numerous simulators are focused on a given specific aspect (*e.g.* terramechanics to study locomotion, contact forces to study manipulation, realistic photo-rendering to study vision algorithms, ...).

The MORSE simulator presented here is not targeted to the study of a specific domain of robotics, but is rather meant as a tool that can be used to test and evaluate *integrated robot software*, *i.e.* the suite of processes required to autonomously achieve complex missions. Using Blender to simulate photo-realistic 3D worlds and the associated physics engine, it brings enough realism to evaluate complete sets of software components within a wide range of application contexts.

The simulator is designed to be useful under varied environments, and provides features to adapt itself to existing robotic architectures, without imposing any changes on them to connect with MORSE. It has modular components that are versatile and configurable, to permit *software-in-the-loop* testing of robotics software. Input from several researchers has guided the evolution of this tool to satisfy the requirements of different labs, while keeping to its design principles. MORSE works on all major software platforms: win32, Linux and OS X. It is developed as an open–source project with a BSD 3-clause license. [6]

The outline of this paper is as follows: Section 2 reviews other comparable robotics simulators, and Section 3 introduces the main design principles of MORSE. Section 4 is the heart part of the article: it depicts the main features of MORSE, each being exemplified with the projects for which they were developed. Finally Section 5 gives the conclusion and future plans for the simulator.

2 Related Work

The Player/Stage/Gazebo[1] is a well known robotics suite. It is a full set of tools that include the Player communication layer and two integrated simulators: Stage [2] is basically a 2D simulator optimized for navigation on flat and closed environments. One of its advantages is the capability to handle very large numbers of simplified robots [3]. However, it is not ideal for more complex scenarios, specially in 3D space. The more recent Gazebo [4] was developed to cover these shortcomings. It has received an important boost in its development with its integration into the ROS platform, and has thus become the most commonly used robotics simulator. It integrates very well with ROS and Player, but connectivity with other middlewares requires additional programming.

The Unified System for Automation and Robot Simulation (USARSim)[5] shares many design concepts with MORSE. It was initially developed as a simu-

[6] User documentation and additional information are available at `http://morse.openrobots.org`, and the source code can be downloaded from `http://github.com/laas/morse.git`.

lator for search and rescue operations, uses the Unreal Engine gaming platform and is built with the concept of modular components. It is widely used as an evaluation tool for the well known RoboCup competitions. However, it uses an adhoc interface to communicate with external software and does not support some of the most common robotics middlewares. The Unreal Development Kit (UDK) used is a closed-source library and is not available on every platform.

Webots [6] is a popular commercial simulator. It provides a full programming environment to create customized robots and environments, but the interface to construct new components and robots is unintuitive and complex, requiring searching trough data trees to adjust physical and geometrical parameters. It also has an integrated code editor, but the control programs created in it must later be converted and transferred to the final robot platform.

Another commercial simulator is V-REP [7], with a large variety of components and sensors available. It allows scripting of the components using the LUA language. Being modular, it allows for copy-and-paste functionality of the components. However, it provides no native method for communication with middlewares; such functionality must be provided by the user in the form of add-ons.

3 MORSE Architecture Overview

The Modular Open Robots Simulation Engine (MORSE) is a library of Python scripts that run on the Blender Game Engine (BGE) to interface a 3D virtual environment with external robotics software. Its main design principles have been described in [8], they are quickly recalled here.

The BGE is used as the base platform for the graphical display of the simulated world and the physics simulation, using the Bullet physics library. The simulation is organized as a main core of control functionality that initializes and coordinates the events in the BGE, and a collection of components that can be used to assemble a robot with sensors and actuators. A variety of communication tools allow each of the MORSE components to connect with external robotics software through middlewares. MORSE has support for some of the most commonly used robotics middlewares nowadays, including ROS [9], Pocolibs [10] and YARP [11]. Other middlewares are regularly being added, mainly driven by the needs of new users. This process is simple, as existing middleware bindings can be used as templates for the new ones.

MORSE sensors and actuators are minimalistic in their functionality and completely middleware agnostic. They can provide similar data as their real world counterparts, but some of them have multiple variants that can work at different levels of realism and abstraction. The data employed by the components can be altered through the use of components called *modifiers*. These add noise or change the data as required to better match the data with real sensors/actuators (for instance, transforming the coordinate frame used by Blender into the Universal Transverse Mercator coordinate system, or adding noise to images captured by Blender cameras). When a simulation scene is created, the components

are linked to middlewares as specified in a configuration script, and the necessary functionality is added to the components to be able to transmit/receive data through the middleware as data streams or synchronous/asynchronous requests. The whole data flow in MORSE can be seen in the diagram of Fig. 1.

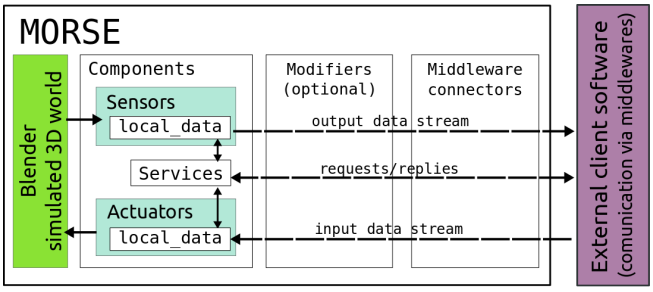

Fig. 1: Overview of the data flow in MORSE, between the simulated world in Blender and the external robotics software

MORSE can be easily extended by adding new components that inherit from the existing base classes. All components are programmed in Python, since this is the language that Blender uses for its scripts. However, when coding components that require a faster processing time, they can be implemented in C/C++ and interfaced with Python using Swig.

4 Morse Main Features

4.1 Blender Integration

Being based on the Blender 3D [12] modeling software, any object, robot or environment can be created and immediately be made ready to use in MORSE. Many file formats for 3D models can be imported directly into Blender, further increasing the range of elements that can be used. The high level of graphical detail is specially useful for realistic looking simulations, and very important when doing image processing from simulated video cameras. It is convenient to have simulation environments recreating the terrain and buildings of real experimentation sites. Transforming the 2D map of a building into a 3D model is straightforward in Blender. However, outdoor terrains for experiments in *field robotics* are more difficult to create by hand. We have developed additional plug-ins for Blender that read Digital Elevation Map (DEM) files and create the appropriate 3D meshes. Adding normal maps and aerial photography for textures creates very realistic simulated environments. Figure 2 shows the modeling capabilities of Blender exploited in MORSE to create robots and environments from DEM data.

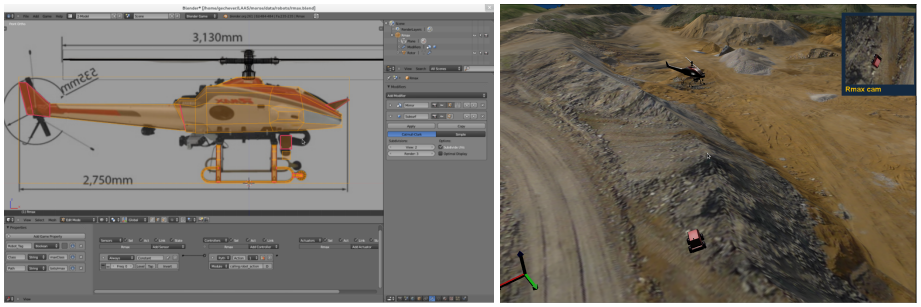

Fig. 2: Left: Modeling of an R-Max robotic helicopter. Right: Robots on a terrain imported from a Digital Elevation Map; the view from a camera mounted on a helicopter robot is shown in the top right frame

4.2 Component Library and Scene Construction

MORSE provides a collection of components that can be used to construct robots with various configurations. *Sensors* recover data from the simulated world and store it. This is done using the Blender predefined interfaces for the BGE and Python scripts. For example, GPS read the absolute coordinates of the sensor in the Blender world, laser scanners use ray tracing operations of z-buffers to generate range images and cameras use renders from the perspective of the Blender camera to produce images. Parameters that determine how the sensors gather data can be adjusted, such as range, resolution, image sizes, etc.

Actuators carry out actions within the simulated world. They provide motion to the robots through several methods and algorithms, or control moving parts such as robotic arms or pan-tilt units. In most cases, the motions are implemented using the functions in Blender to add a certain linear and angular speed to an object. Robotic arms consist of a series of articulated segments, and use the Blender armature system to determine the degrees of freedom at each joint. They can be operated either by direct control of the angles at each joint or by control of the end effector with Inverse Kinematics (using iTaSC [13], a constraints-based IK solver available for the BGE).

Robots are the platform upon which all sensors and actuators must be mounted in order to function. They currently have no behavior of their own, but define the physical shape and properties like mass, friction and collision bounds used by the Bullet physics library to compute the interactions with the environment.

The component library also includes other objects that can be used in the simulation, from large outdoor environments, to furniture and small items. All of these can be imported from individual Blender files into a simulation scenario.

Building and configuration of scenarios are done with a set of Python classes provided by MORSE, known as the *Builder API*. The Builder API provides an internal domain-specific language (DSL) that completely hides the somewhat complex interface of Blender from the user, so that those unfamiliar with Blender can directly configure MORSE using Python scripts. Scripts can then

be tracked on a version control system to follow changes and apply patches. The API offers classes to add robots, sensors and actuators, to position, rename and configure any of the parameters used by these components, to include additional objects (furniture, obstacles, etc.) and to add middleware bindings, modifiers and services for each component.

4.3 Adjustable Levels of Realism

Many components come in several versions, which produce more or less realistic behavior, and can be chosen depending on the objective of the simulation and processing that will be applied to the data.

Ground robots can consist of rigid bodies for chassis and wheels using physical constraints to give realistic movement based on the speeds of the wheels. Alternatively, some robots are handled as a single rigid box sliding over the ground when the details of the motion of the robot are not important. For aerial and submarine robots the physics are limited to collision detection, but they are not affected by gravity, and there is no simulation of air or wind currents – but external dynamic models can be linked to MORSE for such purposes. The associated actuators offer the choice between controlling the speeds of wheels independently, controlling the motion of a robot as a whole with linear and angular velocities, giving direct waypoint coordinates for destination or "teleporting" a robot to a desired location.

Some sensors provide almost identical raw data as their real counterparts, which can then be processed to extract relevant information, and later used to make decisions. When the processing of data is not of interest to the experiment, alternative sensors can be used that provide higher level data, extracted directly from the BGE scenario, and avoid costly processing. The clearest example of this are the cameras: the regular video camera generates renders of the Blender scene, and outputs images to be processed by software. This generates much data, and can slow down the simulation when rendering for several cameras at once. A new sensor called the *semantic camera* uses Blender functions to determine the names and global 3D position of the objects within the view frustum of the camera. It directly outputs this data, making the simulation of object detection faster and avoiding the use of image processing algorithms.

As a practical example, researchers from ONERA use MORSE to evaluate online probabilistic decision-making on an autonomous UAV. In a first scenario [14], MDP-based policy optimization allows the UAV to find a suitable zone for an emergency landing. MORSE video camera sensors are used to feed the raw image data into the land mapping algorithm. On a second scenario [15], a POMDP model is used to optimize the probability to track and intercept an identified target among others. As the objective is to evaluate the optimized strategy, processing simulated images is not necessary. Hence, instead of using a video camera sensor, the *semantic camera* is used to identify the location of the target, adding some errors with modifiers to simulate both the sensor and the detection and identification processes.

4.4 Middleware Configurations

Research laboratories use different middlewares to connect the processes within their robots. In order to be usable with any architecture, MORSE enforces a clear separation between components and middlewares. When a simulation is started, the bindings between sensors/actuators and middlewares are read from the configuration script, and it is only then that components acquire the functions necessary to communicate with the outside world. This separation means that any middleware can be integrated with MORSE, by simply providing the scripts to marshal MORSE data in the expected format. Middlewares currently supported include ROS, MOOS, Pocolibs, YARP and raw TCP/IP sockets. Information exchange between the simulator and the robotics software can be done by a constant data stream, or via request/reply interfaces.

MORSE permits using different middlewares within a single simulation scenario with heterogeneous systems. This is exemplified in the *Action project* [16] developed by the LAAS and ONERA labs. It consists in the cooperation of ground and air robots to patrol a zone, locate and follow moving targets. Each lab provides an existing robotics platform with two different architectures: LAAS employs Segway RMP 400 land robots, using the G^{en}oM [17] architecture, while ONERA works with Yamaha R-Max helicopters based on OROCOS [18]. The simulation uses Pocolibs to communicate with G^{en}oM , and YARP to talk with OROCOS.

MORSE has been chosen as the simulator platform for a French research robotic project called *ANR-PROTEUS*, thanks to its middlewares versatility and the facility to integrate it with existing architectures. The PROTEUS project supports a model-driven engineering approach based on a domain-specific language and a robotic ontology [19] and aims at providing a toolchain for robotic development from modeling to software simulation and deployment on real robots. PROTEUS is based on open-source softwares like the Eclipse Modeling Project, the ROS middleware and MORSE simulator. Several robot models are currently under construction (Wifibot, ECA Camelon, and Thales R-Trooper).

4.5 Component Overlays for Specific Software Architectures

Besides the components and middlewares available, it is further possible to adapt these elements to better fit an existing architecture and allow true *software-in-the-loop* functionality. MORSE components provide dedicated I/O interfaces. In many cases, the interface methods will not be the same as those used in an actual device, although the component has the same functionality. For instance, an actuator may have two separate functions to modify its linear and angular velocities, while the corresponding MORSE component uses only one function with the two parameters.

To avoid creating additional components, MORSE implements the notion of *component overlays*. Following the well-known adapter pattern, they are implemented as special components that override (or wrap) the default behavior to

make it match a different architecture. In the case of middlewares, it is possible to either use the default serialization methods provided by MORSE, or write additional serialization functions as extra scripts, specific for single components. Overlays are implemented as Python scripts that provide an interface between the real component being simulated and the equivalent sensor available in MORSE. These features have been used to connect MORSE with an existing Unmanned Aerial Vehicle (AUV) control architecture [20] without modifying the code in either the original architecture or MORSE.

4.6 Multi-node and Hybrid Simulation

The simulation of multiple robots in the complex environments permitted by MORSE is very demanding on computational and graphics resources. A scenario with several robots equipped with video cameras and other sensors, plus the robotics software all running on the same computer will slow down the system considerably. MORSE offers the possibility of running multiple instances of the same simulation scenario in separate computers, coordinated by a central server program, called the *multi-node manager*. This manager has been implemented using TCP/IP socket communication, and also using High Level Architecture (HLA) [21] for more strict time management. While every node shows all of the robots, each node will only be in charge of controlling a limited number of them. The movements of robots and specific objects in one node are sent to the multi-node manager, which in turn collects the updated positions across all nodes and redistributes the information, so that all nodes can immediately reflect the changes. The multi-node server also synchronizes the time and events across all nodes. This can be used to slow down the simulation in all nodes, by making them wait for the synchronization message. However, at the current time it is not possible to accelerate the simulation speed from the multi-node server.

This functionality was developed for applications that require a large amount of robots. Our use case is the rescue robotics project *Rosace* [22]. Its main objective is the coordination of robotic agents in search and rescue operations in the case of a disaster. In this scenario, terrestrial robots must be able to locate human victims, provide support for the victims and avoid dangerous areas. The robots in the team are to be equipped with different payloads, and take autonomous decisions on which of them should perform different tasks in the mission, such as searching, providing a communications relay, and helping victims directly. For this project, two or three robots can be handled by a single simulation node, and synchronized with those in other nodes. Special sensors in MORSE are used to determine the distance and visibility between robots, and this information is used to simulate loss of connectivity between them. The victims to be saved are internally considered as robots with scripted behavior, so that their status and position is also synchronized by the multi-node manager.

Additionally, the multi-node system permits the deployment of hybrid simulations. A 3D environment that closely represents the real experimentation site is used. Real robots report their updated positions to the multi-node server, which reflects the changes on a dummy robot in the simulation, so that other

simulated robots can see it and interact with it. In the Action project, a real ground robot moves around while the simulated helicopter can follow it using video cameras. The robots communicate in the same way in full simulation or in real life. The use of HLA makes it possible to synchronize the heterogeneous systems in both the simulated and the real world. Note however that the real robots are not able to "see" the simulated ones with their cameras – but this could be achieved using an augmented reality scheme that would modify the data gathered by the real robots.

4.7 The Human Avatar

For human-robot-interaction scenarios, we require a way to combine the reactive and sometimes unpredictable behavior of a human interacting with its environment with a simulated robot. Therefore the human avatar of the MORSE simulator has been equipped with an intuitive control that enables users to act upon the simulated environment. Inspired by modern 3D-computer games, the user takes a third-person perspective behind the human avatar to move around as shown in Fig. 3 (left). While moving around, the camera tries to avoid the objects and walls placed between the camera and the human avatar to prevent occlusions. All objects that can be interacted with can be displayed by pressing a key on the keyboard (also illustrated in Fig. 3). When the user decides to interact with an object, the camera switches to a first-person perspective and offers an interface showing possible actions the user can take when pointing to specific objects, as shown in Fig. 3 (right). Those actions at the moment include picking up and releasing objects, opening and closing drawers and cupboards and switching on and off specific objects like a light or an oven.

Fig. 3: Left: third-person view of the human component that is used to navigate in the environment, displaying the names of objects that the human can interact with. Right: first-person perspective of the human avatar that indicates a possible "pick-up-action" with the bread.

The motions of the avatar are animated using Blender armatures, inverse kinematics, and predefined movement loops. The avatar can be controlled much like a character in a videogame, using either the mouse and keyboard or a combination of the Microsoft Kinect and the Nintendo Wiimote. This enables users to perform pick and place actions in simulated worlds while at the same time the simulated robot(s) can be controlled through the supported robotic middlewares.

The human avatar is meant to be used in *personal robotics* scenarios. In these, complex robots are expected to collaborate with humans to carry out ordinary household tasks, such as cleaning, serving food or aiding humans to navigate an environment. Robots used in these experiments are equipped with one or two arms, and are capable of grasping objects. They are also expected to react to the actions of their human collaborator, using video cameras, motion detectors or telemetry to determine the location, pose and attitude of the human. This can be done at two different levels of abstraction. Using the video cameras to recognize the human and its pose can be done realistically, with the associated computational cost and uncertainties. Alternatively, the avatar can directly export the position of all of its joints, and feed them back to the robot, simulating a full motion capture system and avoiding the processing costs.

An example use case in this scenario is the testing and validation of human-aware navigation planners of service robots in human-centered environments at TU Munich. In this case, a simulated human tracking system provides the human pose to the robot while the robot navigates in the environment in a way that is safe and legible for the human [23]. In this project, MORSE has not only been used as a powerful tool for testing human-aware navigation strategies before carrying out experiments with real humans: it has also been used to evaluate them as Lichtenthäler et al. show in [24] by video-based user studies. After successful testing and evaluation, the human-aware navigation was applied using real robots and humans resulting in a safe and reliable behavior of the robot.

5 Summary and Future Work

We have presented the main features developed within the MORSE simulator, following the requirements of a variety of projects in robotics research. The design and architecture of MORSE has proved to be flexible and powerful enough to allow researchers to use it under various circumstances to test their robotics algorithms. Users can customize the existing components according to their needs, or develop new ones when necessary, by describing their behavior in Python scripts. All new developments done on top of MORSE are made available to the whole community, thanks to the open source license of the simulator.

MORSE allows for quick integration with an existing architecture (multiple middlewares, modular components and component overlays), heterogeneous robots (mixing components and middlewares), multiple robot simulation (multi-node synchronization) and human-robot interaction (high abstraction level sensors and human avatar).

Further work is planned to increase the usefulness of MORSE: for human-robot-interaction experiments, it is ideal to have a deeper immersion when using the human avatar. A planned feature is to provide stereo images of the simulation to the user wearing special goggles. This can trivially be done from Blender, with some separation of the images produced for the two eyes. Increased integration with motion reading devices, such as Microsoft Kinect can also make the experience more natural, by allowing a finer control over the human avatar. Another planned work is to be able to couple MORSE simulations with other physical simulation engines thanks to the HLA support. Further development is also necessary to give users more control over the speed of the simulation, by adjusting the base execution frequency of the BGE.

Acknowledgments This work has been partially supported by the DGA funded Action project (`http://action.onera.fr`), the STAE foundation Rosace project (`http://www.fondation-stae.net`) and the ANR-PROTEUS project (`http://anr-proteus.fr/`)

References

1. R. B. Rusu, A. Maldonado, M. Beetz, and B. P. Gerkey, "Extending Player/Stage/Gazebo towards cognitive robots acting in ubiquitous sensor-equipped environments," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) Workshop for Network Robot Systems*, (Rome, Italy), 2007.
2. B. P. Gerkey, R. T. Vaughan, and A. Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems," in *Proceedings of the 11th International Conference on Advanced Robotics*, pp. 317–323, 2003.
3. R. Vaughan, "Massively Multi-robot Simulation in Stage," *Swarm Intelligence*, vol. 2, pp. 189–208, 2008.
4. N. Koenig and A. Howard, "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator," in *In IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2149–2154, 2004.
5. S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "USARSim: a robot simulator for research and education," in *Proceedings of the 2007 IEEE Conference on Robotics and Automation*, pp. 1400–1405, April 2007.
6. O. Michel, "Webots: Professional mobile robot simulation," *Journal of Advanced Robotics Systems*, vol. 1, no. 1, pp. 39–42, 2004.
7. M. Freese, S. Singh, F. Ozaki, and N. Matsuhira, "Virtual robot experimentation platform v-rep: a versatile 3d robot simulator," in *Proceedings of the Second international conference on Simulation, modeling, and programming for autonomous robots*, SIMPAR'10, (Berlin, Heidelberg), pp. 51–62, Springer-Verlag, 2010.
8. G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular open robots simulation engine: Morse," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 46 –51, May 2011.
9. M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

10. "Pocolibs: Posix communication library." `http://pocolibs.openrobots.org`.
11. G. Metta, P. Fitzpatrick, and L. Natale, "YARP: yet another robot platform," *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, pp. 43–48, 2006.
12. "Blender 3D." `http://www.blender.org/`.
13. R. Smits, T. De Laet, K. Claes, H. Bruyninckx, and J. De Schutter, "iTASC: a tool for multi-sensor integration in robot manipulation," in *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pp. 426 –433, August 2008.
14. F. Teichteil-Königsbuch, C. Lesire, and G. Infantes, "A Generic Framework for Anytime Execution-driven Planning in Robotics," in *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA)*, (Shanghai, China), pp. 299–304, 2011.
15. C. P. Carvalho Chanel, F. Teichteil-Königsbuch, and C. Lesire, "POMDP-based online target detection and recognition for autonomous UAVs," in *Proceedings of the 7th Conference on Prestigious Applications of Intelligent Systems (PAIS)*, (Montpellier, France), 2012.
16. R. Boumghar, S. Lacroix, and O. Lefebvre, "An information-driven navigation strategy for autonomous navigation in unknown environments," in *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on*, pp. 172 – 177, November 2011.
17. A. Mallet and M. Herrb, "Recent developments of the GenoM robotic component generator," in *6th National Conference on Control Architectures of Robots*, (Grenoble, France), INRIA Grenoble Rhône-Alpes, May 2011.
18. H. Bruyninckx, P. Soetens, and B. Koninckx, "The Real-Time Motion Control Core of the Orocos Project," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2766–2271, September 2003.
19. S. Dhouib, N. du Lac, J.-L. Farges, S. Gerard, M. Hemaissia-Jeannin, J. Lahera-Perez, S. Millet, B. Patin, and S. Stinckwich., "Control architecture concepts and properties of an ontology devoted to exchanges in mobile robotics.," in *Proceedings of the 6th National Conference on "Control Architecture for Robots"*, 2011.
20. M. Barbier, J.-F. Gabard, A. Bertholom, and Y. Dupas, "An Onboard Software Decisional Architecture for Rapid Environmental Assessment Missions," in *Proceedings of the 18th IFAC World Congress*, (Milano, Italy), pp. 11797–11802, 2011.
21. F. Kuhl, R. Weatherly, and J. Dahmann, *Creating computer simulation systems: an introduction to the high level architecture*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999.
22. J. Lacouture, J. Gascueña, M.-P. Gleizes, P. Glize, F. Garijo, and A. Fernández-Caballero, "Rosace: Agent-based systems for dynamic task allocation in crisis management," in *Advances on Practical Applications of Agents and Multi-Agent Systems* (Y. Demazeau, J. P. Müller, J. M. C. Rodríguez, and J. B. Pérez, eds.), vol. 155 of *Advances in Intelligent and Soft Computing*, pp. 255–259, Springer Berlin / Heidelberg, 2012.
23. T. Kruse, A. Kirsch, E. A. Sisbot, and R. Alami, "Exploiting human cooperation in human-centered robot navigation," in *IEEE International Symposium in Robot and Human Interactive Communication (Ro-Man)*, 2010.
24. C. Lichtenthaeler, T. Lorenz, M. Karg, and A. Kirsch, "Increasing Perceived Value Between Human and Robots - Measuring Legibility in Human Aware Navigation," in *IEEE workshop on Advanced Robotics and its Social Impacts*, 2012.