ASCARI: a component based simulator for distributed mobile robot systems

Mirko Ferrati, Centro di Ricerca "E.Piaggio", Dipartimento di Ingegneria dell'Informazione, Università di Pisa, mirko.ferrati@gmail.com, Italy

Alessandro Settimi, Centro di Ricerca "E.Piaggio", Dipartimento di Ingegneria dell'Informazione, Università di Pisa, alessandro.settimi@for.unipi.it, Italy

Lucia Pallottino, Centro di Ricerca "E.Piaggio", Dipartimento di Ingegneria dell'Informazione, Università di Pisa, l.pallottino@centropiaggio.unipi.it, Italy

Abstract

ASCARI is a simulator dedicated to distributed and cooperative mobile robotics systems. *ASCARI* has been designed to be a generic framework for implementing and testing multi-agent collaborative algorithms, especially suited to evaluate algorithms performances with a non-perfect communication channel (e.g. delayed, limited bandwidth, limited range). The design process has taken into account state-of-art robotics simulators but was mainly driven by a complex new requirement: inter-agent communication has to be integrated in the simulation loop.
The core of the project is a server with a simple dynamic engine, a synchronization facility and an API (Application Programming Interface) to control simulated communication which is used by server plugins to provide implementations for the characteristics of the communication channel. Different channel requirements can be added by users through the dedicated plugin. Beside the server, there are the agents involved in the simulation.The only custom code required from the user is the agent control law.
The control law has to be written as a plugin which receives sensors data and control actuators using an API as an abstraction layer from hardware. As in every mature simulator, the abstraction layer quickly enables the developer to use the desired control law directly on a real robot by changing only hardware drivers.
Finally, the inter-agent communication is provided in a transparent way to the user by some template communication classes. The user can exchange information between agents by simply creating senders and receivers classes with their custom data type. Simulated communications and filters are all handled automatically. The simulator is completed by a 2D viewer and a simple GUI (Graphical User Interface) that allow the user to intuitively follow the simulation evolution and to start the various processes (simulator, agents, viewer), controlling the number of agents involved in and the simulation speed.
In this paper we first describe the *ASCARI* simulator and we then validate it on a distributed traffic control and a distributed task assignment algorithm.

1 Introduction

Simulations are widely used in multi-robot systems as a validation method, yet most of the software simulators are usually developed to target a specific application/algorithm, thus they lack the capabilities to become a general and stardardized tool for researchers in this field. While, for generic robotics applications, there exist some simulators that are recognized as standard tools, these softwares are not very suited to multi-robot systems. Here we will briefly examine the features of some of the most known robotic simulators from the point of view of multi-agent applications. Specifically, our aim is to build a system that allows easy testing of communication-based distributed algorithms, where the communication channel and equipment can be specified by the user. The most inspiring simulator for our work was Stage, from the project Player/Stage/Gazebo. The Player/Stage (P/S) [1] project is an open-source software for rapid development of robot control code. Player [3, 2] is based on a set of simple interfaces for communicating and controlling sensors and actuators, while Stage is a 2D simulator for multiple robots that includes a simple physical simulations. Stage uses a set of simulated devices that are compatible with standard Player interfaces. Thus the interface that is presented to the user by Player can be used unchanged from simulation to real hardware.

For example, a program that drives a simulated laser-equipped robot in Stage will also drive a real laser-equipped robot, with no changes to the control code.

Stage software architecture was designed with multi-robot systems in mind, the resulting software is capable of handling thousands of mobile agents with simple dynamics and an arbitrary control code. The main limitation of P/S is the communication between robots: it was not designed for simulating a communication, and the Wifi interface has been only declared as a possible sensor, but it was not implemented.

We believe that not taking into account the capability of simulating communications among agents during the design of Stage makes it very hard to add that feature now. Anyway, the project is no longer developed since some years, maybe due to the focus of open source robotics community on Gazebo. Gazebo has recently evolved as a standalone project integrated with ROS, and lost its old Player interface coupling. The main difference between Gazebo and Stage is the quality and the precision of the dynamic integration: Gazebo is a 3D environment with accurate body collisions and joint simulations, focused on a single robot hardly interacting with the environment. Since one year Gazebo provides an 802 wireless simulated sensor with Hata-Okumara propagation model. This sensor can be used as a starting point for developing other type of wireless communication sensors. Gazebo is not suited for thousands of robots because of the computational complexity of the simulation: simulating thousands of robots in Gazebo would require too much time and usually researchers in distributed control algorithms for multi-robot systems are more interested in fast approximated simulations than in realistic ones. The same considerations stand for similar simulators such as V-rep [9], USARSim [8] and Microsoft Robotics Studio [7].

MatSim [4] uses a totally different approach to multi-agent simulations: by ignoring agent dynamics it is capable of simulating more than 1 millions of agents in the same environment. It is a grown and stable software easy to use, but as such, it appears complex to understand and edit in its simulating capabilities.

Webots [5] is a commercial closed-source 3D mobile robot simulator. The robot controllers can be programmed with the built-in IDE (Integrated Development Environment) or with third party development environments. The robot behavior can be tested in physically realistic worlds. The controller programs can optionally be transferred to commercially available real robots. A simulated wifi-communication module is implemented inside the simulator but, since Webots is not expandable by the user, the communication model cannot be changed or improved.

Argos [6] was inspired by Stage. The main differences can be found in the simulation speed (Argos is faster) and in the decoupling of the physics engine from the simulated world. Argos suffers the same design problem of Stage related to the possibility of simulating communication channels. The user cannot influence the simulator side, meaning that he/she can't write a communication module integrated with the simulated world. As in Stage, the wifi sensor exists but it has an empty behavior, meaning that it was not implemented.

Mason [10] is a multi-agent simulation suite written in Java to be portable on different operating systems. It was not built as a complete simulator, but as something to be easily adapted by other researchers into a custom simulator. Mason is focused on agents in a wider sense, so it does not provide any mobile robotics facility such as communications, sensors or a realistic dynamic.

It is worth noting Swarm [11], which is probably one of the oldest multi-agent simulators and served as inspiration for Mason, Argos and Stage.

Finally, two new simulators have been growing in the last years: RoboRobo! [12] and Morse [13]. They are both inspired by the need of new fast simulators dedicated to multi-agent research and developed with state-of-the-art technologies. Roborobo! can be pretty useful as a base for developing a custom simulator for a particular application (e.g. a specific swarm mobile robotics environment and simulation), but not as simulator capable of providing a platform for testing many different algorithms. Morse is built with interoperability as a priority: it supports most known middlewares (ROS, YARP) and uses a script to set up simulations. Robot models inside Morse are controlled through interprocess communication, new models can be added by writing sensors and actuators simulation code as python scripts. At the moment of writing, there are no simulated communication sensors in the simulator sensor library.

Although the state of art of simulators is mature and fulfills almost every research requirements, we believe that a generic communication system integrated in the simulation loop is a missing feature. We will propose a design capable of providing this new feature while keeping the best practices from state of art simulators. We believe that our communication system design can successfully be added to newborn simulators such as RoboRobo and Morse.

Our work aims to have a completely application–independent simulator suited for distributed mobile robotics with simulated communication network, which can be used together with simulated robots and real ones, see figure 1.

This paper is organized as follows. In section 2 we describe the requirements which our simulator was designed for. The global software architecture is addressed in section 3, here the communication facilities

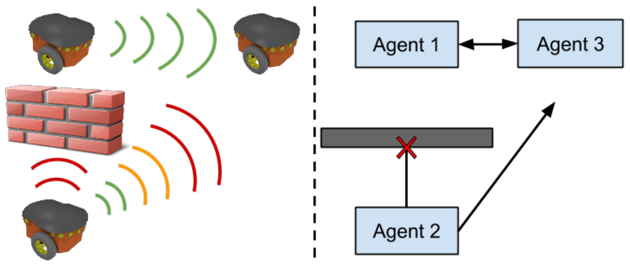


Figure 1: The *ASCARI* simulated communication.

are introduced, a more detailed description is in section 4. The hardware integration is addressed in section 5. We describe two application that use our simulator in section 6.

2 Design requirements

ASCARI has been developed with a set of requirements that were identified as main features from state of art simulators. In fact, we believe that building a simulator primarly around novel features would lead to a custom non-generic software, our target is instead to set a new standard for multi–robot systems simulations (as Stage did in 2008).

One of the major features of modern simulators is the possibility to use the same code for both simulated and real agents. The only changes required should be the input/output drivers, not the control law or its implementation.

A difference between custom simulators and the widespread ones is their modular and expandable design. A simulator should never have hardcoded assumptions on the system that will be simulated, new modules with new features should be easily integrated into the simulator, even core features should be built as modules. Argos and Gazebo fully represents this approach by using interfaces and plugins.

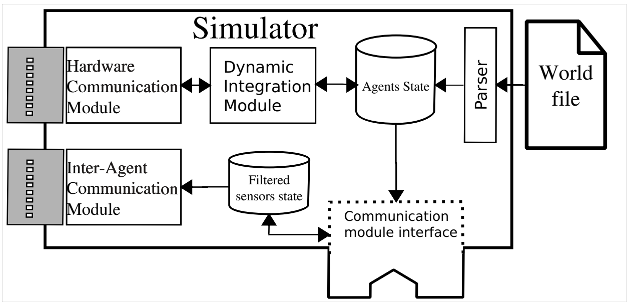
Usually, simulators support multi–process systems where each robot controller runs in its own process or computer, while the simulation handles dynamic integration and model collisions. This approach is a limitation when simulating thousands of robots, and the solution is the capability of integrating the robot controllers directly into the same simulation process, as Stage does. Designing *ASCARI*, we chose a mixed approach where controllers can be wrapped in a standalone process or run in a different thread of the simulator process itself. In the next section we will show how this capability emerged naturally from *ASCARI* software architecture design.

Following the Gazebo plugin approach, *ASCARI* supports any kind of dynamic law required by the user. Dynamic laws can be expressed directly as code or as a set of equations: $\dot{x} = f(x, u)$ where $x \in \mathbb{R}^n$ is the state of the agent, $u \in \mathbb{R}^m$ is the controller output and f is, in general, a non-linear function.

Finally, our main target is the design of a communication system integrated in both the simulation loop and the real robots. The implementation of the communication between agents is based on communication primitives. Such primitives manage any kind of message data and hide the simulator to the user providing the same API for real and simulated agents. The communication module inside the simulator has been designed as a plugin that may be changed to model various communication channels with different characteristics such as ranges, signal noises and visibility.

3 Software Architecture

In this section we describe some of the main *ASCARI* software components, focusing in particular on the design choices related to the communication facilities. From an high level point of view, *ASCARI* has a server that acts as a world simulator, a set of clients (simulated agents) and a simple 2D/3D visualizer.



Simulator
Hardware Communication Module
Dynamic Integration Module
Agents State
Parser
World file
Inter-Agent Communication Module
Filtered sensors state
Communication module interface

Figure 2: The *ASCARI* server architecture.

The important server sub-modules are shown in figure 2. The server parses a configuration file where the world and the agents are described. A database with each agent state is then created and updated by a dynamic integration module at each simulation step after receiving commands from clients. A user code is loaded as a communication module, it has reading access to the agent state database, and updates an agent communication filter database, which is used to allow or block inter-agent communications. At the moment, two dynamic modules are available, a simple fixed step forward integration (default) and an external robot state receiver, which can receive updated states from any external dynamic integration system or real hardware sensors, allowing hardware in the loop simulations (see details in section 5). The server basic loop flowchart is shown in figure 3.

It sends the updated state to each agent, receives new actuator controls from them, integrates the dynamic, executes the communication module update function, forwards all the inter-agent communications, based on the agent communication filter database, and starts again.

The agent structure is represented in figure 4. As already said, *ASCARI* is based on many little building

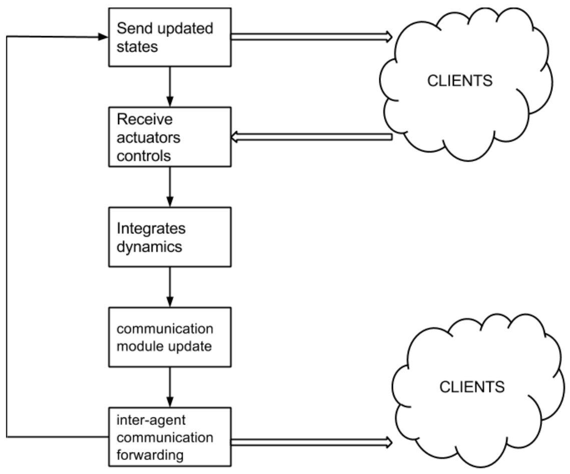

Send updated states
Receive actuators controls
Integrates dynamics
communication module update
inter-agent communication forwarding
CLIENTS
CLIENTS

Figure 3: The *ASCARI* server flowchart.

blocks, so many client sub-modules are similar to their respective ones in the server. Hardware and inter–agent communication module are connected to the server modules. The configuration file is read by the parser and it is used to provide initialization parameters to the agent control loop.

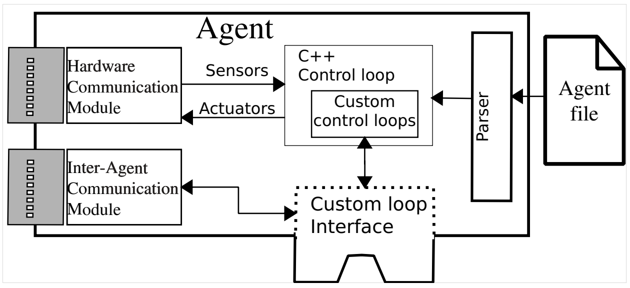


Figure 4: The *ASCARI* client architecture.

The control loop handles sensors and actuators, providing the data to the custom control loop as it was local and not remotely simulated. The custom loop interface is where user code is running. It has access to the agent state and to the inter-agent communication primitives that allows the user to create distributed algorithms. The parser results are given also to the custom loop, so that user can specify additional parameters in the same agent configuration file. The client basic loop flowchart is shown in figure 5.

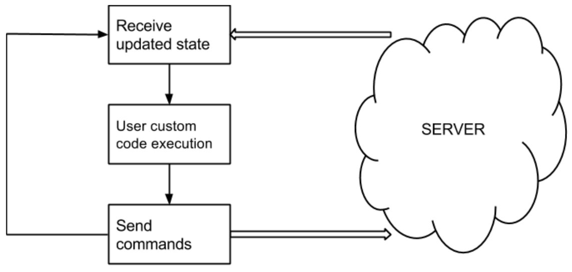


Figure 5: The *ASCARI* client flowchart.

It receives the updated state from the server, executes the user custom code, and sends the resulting commands to the simulator.

Configuration file structure and options

An example of a configuration file is reported. The structure is the classic one of the yaml files. The user can use the provided parser to get the information from the file in his/hers custom code.

Inside the tag *WORLD* (where we can define additional variables) there are the two main tags: *BEHAVIORS* and *AGENTS*.

```
- WORLD:

  BEHAVIORS:
  - name: unicycle
    states: [x,y,theta]
    control_commands: [v,w]
    DYNAMIC_MAP:
    - x: 'v*cos(theta)'
      y: 'v*sin(theta)'
      theta: 'w'

  AGENTS:
  - agent: Agent1
    COMMUNICATION_AREA:   circle(50)
    INITIAL:
    - x: '3'
      y: '3'
      theta: '0'
    BEHAVIOR: unicycle
    SIMULATED: 1
```

Inside the *BEHAVIORS* tag the user can define many agent types with different states and dynamics.

Inside the *AGENTS* tag the user can define as many desired agents with their own parameters and behaviours. In the example, the variable *COMMUNICATION_AREA* is used by a communication filter module to filter all the messages outside a circle centered in the robot and with a radius of 50 meters. The *SIMULATED* tag allows to use the same configuration file for real and simulated agents, which can also be mixed together to obtain an hardware in the loop simulation.

Finally, a 2D/3D visualizer and a simple GUI to start both the server and all the agents specified in a configuration file complete the *ASCARI* software architecture. The visualizer reads the network traffic generated between the server and the clients in a transparent way, so that zero or many viewers can be attached to the same simulation.

4 Communication Facility

The core of our system is a set of communication modules and interfaces organized as generic basic building blocks, usually a sender and a receiver. Both the communication between clients and server and inter-agent communication modules are implemented using these blocks. Messages are strongly typed and can be any complex c++ structure, including any STL container. Receiver and sender are c++ classes template–ized on the type of message they work with, they are initialized with an IP address, as 1-to-1 or 1-to-N connections, and they can have an optional topic name similar to ROS topics used to filter messages. Usually a developer uses both classes and creates a transceiver made with a pair of receiver/sender. The agent code can use the transceiver to communicate with other agents without caring about the presence of a simulator. In fact, if a simulation is running, communications are automatically synchronized with the simulation loop among all the agents, while in the case of real agents, the same transceiver creates direct inter–agent connections, thus the agent code does not require any changes in the communication part when switching from simulated to real hardware. The only changes required are inside the configuration file, by editing the *SIMULATED* tag.

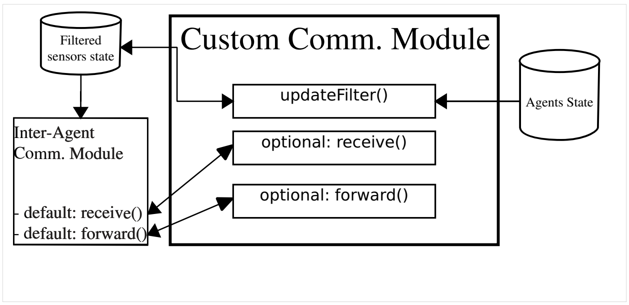


Figure 6: : The *ASCARI* custom Communication Module.

From the simulator point of view, the user needs to inherit a simple *filterModule* that represents a communication channel and to create a custom communication module (figure 6). We chose to give the user maximum freedom on the channel behaviour but, in its simpler form, the inherited class has just to implement an *updateFilter(...)* method. The *updateFilter* inputs are the simulation states of all the agents, defined as states variables (e.g. planar coordinates of agents). This method updates a filtering table that tells the simulator if two agents are able to communicate to each other at each simulation step. If the user wants more control, he can also override the *receive(...)* and *forward(...)* methods. These two methods are the ones used by the server in order to receive and transmit inter–agent messages by acting as a gateway. If they are reimplemented, advanced behavior such as delays on the channel, random packets drop, bandwidth limitations and so on can be obtained. Note that *receive* and *forward* methods have access to the messages content, while the *updateFilter* has, as input, only the agents state.

5 Hardware Integration

All the code written in *ASCARI* can be reused on the real hardware thanks to the communication facility. During the execution, if in the configuration file an agent has been declared as not simulated, instead of sending the commands to the simulator, these are sent via serial communication directly to the hardware through a serial communication block that fully complies with the standard messages that the client/server exchanges during the simulation. Basically instead of using the actuation commands to simulate the dynamics of the robot, these are used to move the real robot. With this configuration the only thing a user, that want to pass to the hardware, has to care about is to have something that can receive via serial bus the actuation commands and execute them. The commands for the robots currently used are *linear acceleration* and *angular acceleration*.

An example of the hardware we used is shown in figure 7.

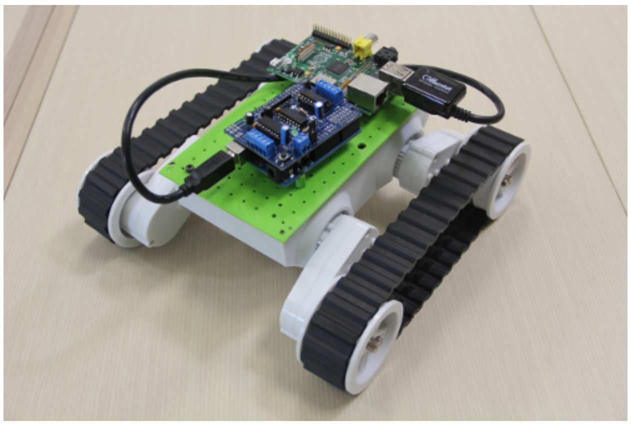


Figure 7: A mobile robot, equipped with a *Raspberry Pi* and an *Arduino Uno*

In this robot there is a *Raspberry Pi*, which is the core of the robot. On board there are a wireless key to communicate with the other agents and a Linux system on which the various algorithms run. Plugged to it via serial bus there is an *Arduino Uno* which receive the actuation commands from the Raspberry and send them to the motors via a shield board (wires have been omitted for the sake of clarity).

One advantage to have this architecture is that we can work with real and simulated robot as well just defining the type of the robots in the configuration file. For example we can run a collision avoidance algorithm on two real robot and a simulated one. In the real scenario we will see the real robots steering 'near' the simulated one, even if there is nothing in front of them.

6 Applications

Many applications have been developed thanks to *ASCARI*, both on the software and on the hardware side. These applications concern for example: collision avoidance, agents localization using webcams and markers and classical distributed robotics algorithm such as rendez-vous and distributed consensus.

Two of the most interesting applications are a distributed task assignment algorithm and a distributed traffic control one.

In the task assignment application a subgradient method based algorithm allows to assign various tasks to heterogeneous robots optimally, [15]. Concerning *ASCARI*, it has been used to simulated the various robots' dynamics and to coordinate them during the scenario evolution. Thanks to the communication facility the agents exchange through the network their local subgradient, allowing the other robots to compute the total subgradient until the algorithm converges. In the simulated scenario, the simulator takes into account the dynamicity of the tasks, making them appearing, evolving and expiring during time. The tasks' state is sent through the network by the simulator, so the various agents can know if a task is already taken, or expired, or if some new tasks are available. In the real scenario the tasks are the classical industrial ones, such as putting material on a conveyor belt, or move pallets across a warehouse. In this application the robot recharging is seen has a high priority task, depending on the amount of charge of the robot, which appears if the amount of a robot's charge goes below a certain value. In Figure 8 a frame of a simulation of the algorithm in an industrial scenario is depicted.

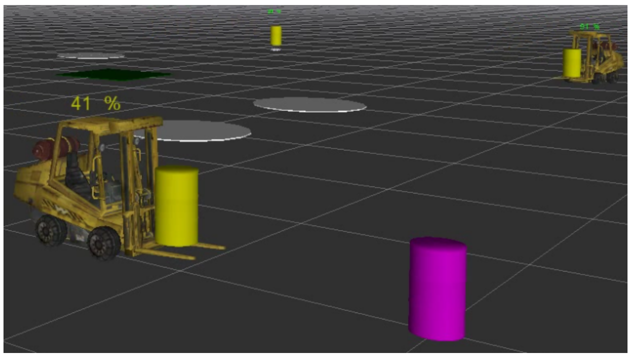


Figure 8: Distributed task assignment algorithm in *ASCARI*.

In [14], a distributed traffic control protocol has been proposed. The algorithm uses communication to foresee and avoid agents collisions on a topological map represented with a graph (see Figure 9). The messages exchanged by agents are a set of future nodes and arcs that each agent would like to cross, a resource locking algorithm ensures that agents will not try to cross the same node in the same future time and hence ensure the safety of the system. The algorithm was tested with different communication ranges and with random packets drop to assess its robustness. In the single-process version of *ASCARI* 30 agents with a simple unicycle dynamics were simulated at around 50x real time with an integration step of 0.001 s.

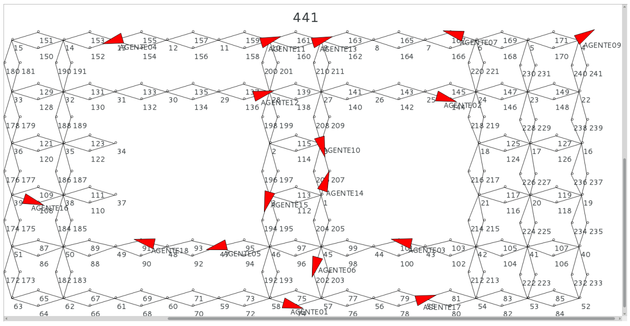


Figure 9: Distributed traffic control algorithm in *ASCARI*.

Notice that *ASCARI* can be used to implement and test any robot control algorithm, but it is especially suited for multi–agent collaborative algorithms also with a non–perfect communication channel. A user must define the agents' control law and a dedicated communicator if particular data exchange between the agents (or between the agents and the simulator) is needed.

To view the original videos please go to the ASCARISimulator channel on:

http://www.youtube.com/user/ASCARIsimulator.

7 Conclusions

In this paper the *ASCARI* distributed mobile robotics simulator with simulated communication network has been presented. The simulator is suited to develop many applications, and in particular it has been designed to be scalable and hence it is able to handle many robots. The hardware integration is straightforward and the user can use real robots and simulated ones as well. Future works include: export the communication facilities to Argos and RoboRobo! and real object simulation (as done in the task assignment application).

References

[1] Vaughan, Richard. "Massively multi-robot simulation in stage." Swarm Intelligence 2.2-4 (2008): 189-208.

[2] Vaughan, Richard T., Brian P. Gerkey, and Andrew Howard. "On device abstractions for portable, reusable robot code." Intelligent Robots

and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on. Vol. 3. IEEE, 2003.

[3] Gerkey, Brian, Richard T. Vaughan, and Andrew Howard. "The player/stage project: Tools for multi-robot and distributed sensor systems." Proceedings of the 11th international conference on advanced robotics. Vol. 1. 2003.

[4] Balmer, Michael, et al. "MATSim-T: Architecture and simulation times." Multi-agent systems for traffic and transportation engineering (2009): 57-78.

[5] Michel, Olivier. "Webots: Symbiosis between virtual and real mobile robots." Virtual Worlds. Springer Berlin Heidelberg, 1998.

[6] Pinciroli, Carlo, et al. "ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems." Swarm intelligence 6.4 (2012): 271-295.

[7] Jackson, Jared. "Microsoft robotics studio: A technical introduction." Robotics & Automation Magazine, IEEE 14.4 (2007): 82-87.

[8] Carpin, Stefano, et al. "USARSim: a robot simulator for research and education." Robotics and Automation, 2007 IEEE International Conference on. IEEE, 2007.

[9] Freese, Marc, et al. "Virtual robot experimentation platform V-REP: a versatile 3D robot simulator." Simulation, Modeling, and Programming for Autonomous Robots. Springer Berlin Heidelberg, 2010. 51-62.

[10] Luke, Sean, et al. "MASON: A Java multi-agent simulation library." Proceedings of Agent 2003 Conference on Challenges in Social Simulation. Vol. 9. 2003.

[11] Minar, Nelson, et al. "The swarm simulation system: a toolkit for building multi-agent systems." Santa Fe NM: Santa Fe Institute Working Paper (1996): 96-06.

[12] Bredeche, Nicolas, et al. "Roborobo! a fast robot simulator for swarm and collective robotics." arXiv preprint arXiv:1304.2888 (2013).

[13] Echeverria, Gilberto, et al. "Modular open robots simulation engine: Morse." Robotics and Automation (ICRA), 2011 IEEE International Conference on. IEEE, 2011.

[14] M. Ferrati and Pallottino, L., "A time expanded network based algorithm for safe and efficient distributed multi-agent coordination", in IEEE Conference on Decision and Control, Florence, Italy, 2013, pp. 2805 - 2810.

[15] A. Settimi and Pallottino, L., "A Subgradient Based Algorithm for Distributed Task Assignment for Heterogeneous Mobile Robots", in IEEE Conference on Decision and Control, Florence, Italy, 2013, pp. 3665 - 3670.