Chapter IV.2

Adaptation and Awareness in Robot Ensembles: Scenarios and Algorithms*

Carlo Pinciroli[1], Michael Bonani[2], Francesco Mondada[3], and Marco Dorigo[1]

[1] Université Libre de Bruxelles, Belgium
[2] Association Mobsya, Switzerland
[3] École Polytechnique Fédérale de Lausanne, Switzerland

Abstract. This chapter presents a disaster recovery scenario that has been used throughout the ASCENS project as a reference to coordinate the study of distributed algorithms for robot ensembles. We first introduce the main traits and open problems in the design of behaviors for robot ensembles. We then present the scenario, highlighting its generality as a framework to compare algorithms and methodologies for distributed robotics. Subsequently, we summarize the main results of the research conducted in ASCENS that used the scenario. Finally, we describe an example algorithm that solves a selected problem in the scenario. The algorithm demonstrates how awareness at the ensemble level can be obtained without requiring awareness at the individual level.

Keywords: swarm robotics, mobile robotics, autonomous robotics

1 Introduction

Large multi-robot systems (*robot swarms*) [2] have the potential to display desirable properties, such as robustness to individual failures through redundancy, and enhanced performance through parallelism and cooperation [11,20]. Realizing such potential is challenging because of the lack of sound design methodologies [5].

In the literature, coordination among multiple robots has been achieved in several ways. Existing approaches span from complete centralization to complete decentralization, with hybrid centralized-decentralized systems in between. With complete centralization, a master system must collect the data from the robots, analyze it and send the actions to perform to each robot. In many applications, the advantages of this approach do not counterbalance its drawbacks. Although centralized control is usually simpler to design and can result in a globally optimized behavior, it suffers from poor robustness (the master system is a single point of failure) and poor scalability (the master system's CPU and network connectivity are shared resources), and it requires global sensing and communication (which is not always available).

* This research was supported by the European project IP 257414 (ASCENS).

M. Wirsing et al. (eds.): Collective Autonomic Systems, LNCS 8998, pp. 471–494, 2015.

In contrast, completely distributed coordination algorithms do not exploit any kind of master system, global knowledge, or planning. Instead, coordination is the result of the parallel pairwise interactions of the system's components. Completely distributed coordination algorithms achieve scalability through local sensing and communication, and achieve robustness and high performance by leveraging the natural parallelism and redundancy of the system. However, it is very hard to design effective coordination algorithms of this kind [10].

To date, the design of swarm robotics systems follows two general types of approaches: behavior-based and automatic methods. Behavior-based methods [1] are typically bottom-up design methods whereby the designer gradually refines the individual robot behaviors until the desired global (i.e., ensemble-level) behavior is achieved. The results obtained with behavior-based methods strongly depend on the experience and ingenuity of the designer. The lack of methodologies above mentioned is partially circumvented by taking inspiration from models of biological systems that display some form of *swarm intelligence* [3,13], such as colonies of ants, bees and termites. However, the complexity currently achieved by these methods is limited, and very far from that of the natural models which inspire the design.

In automatic methods, such as reinforcement learning [36] evolutionary robotics [28], and optimization-based approaches [15], the individual robot behavior is regulated by a set of parameters that are set by a suitable algorithm. These methods allow the designer to focus efforts more on the task to solve, rather than on the individual robot behavior. However, the performance of these methods is known to scale poorly with the complexity of the task to solve and of the robot interactions.

A promising approach to the design of swarm robotics systems is a combination of behavior-based (compositional, pattern-based) aspects and automatic procedures (not restricted to optimization methods). The work in the ASCENS project followed the line of research that leads to the definition of such a combined approach.

In this chapter, we describe the research activities we conducted to apply the ASCENS concepts to state-of-the-art problems in swarm robotics. These activities involved two primary tasks:

1. The definition of a class of application scenarios that provides sufficient complexity to motivate the ASCENS research;
2. The development of algorithms that solve selected problems in the application scenario, in order to nurture and showcase ASCENS techniques and tools.

This chapter is structured as follows. In Section 2, we discuss the mapping of the concept of service component ensemble to robot swarms, introduce the robotic platform employed for experimentation, and present the scenario and its variants. In Section 3, we discuss awareness and adaptation in robot swarms, illustrating the work we made throughout the project. In Section 4 we present two algorithms that demonstrate some of the concepts studied in ASCENS. We conclude

the paper in Section 5, summarizing our work and proposing ideas for future investigation.

2 Scenario: Disaster Recovery

In this section, we present the application scenario on which we based the robotics case study. In Section 2.1 we discuss the mapping between the concepts of *service component ensemble* and *robot swarms*. In Section 2.2 we present the robotic platform we employed for the work in this case study. In Section 2.3 we provide a general description of the scenario. In Section 2.4 we illustrate the possible variants of the robotics scenario.

2.1 Robot Swarms as Service Component Ensembles

Robots swarms can be cast as *service component ensembles* in several ways, depending on the focus of the designer.

A first approach is to consider a single robot as a service component and robot swarms as service component ensembles. In this case, the design neglects the internals of the robot, which becomes a black box that exposes a set of functionalities. The focus of the design is set on the coordination of the robot swarm as a whole and on the correctness of the individual actions with respect to a common goal.

Alternatively, one might represent a single robot as a distributed system composed of a collection of microprocessors. Each microprocessor is responsible for the control of a subset of the available devices. To achieve coordination, the microprocessors communicate. Under this light, in ASCENS parlance each microprocessor is a service component, and a robot is a service component ensemble. Robot swarms, in turn, become *ensembles of service component ensembles*. Thus, the focus of the design spans two layers: at the lower layer, the design must ensure that each robot device behaves correctly; at the higher layer, the common goal of the swarm must be achieved.

The choice between these two approaches is ultimately dictated by the requirements of the algorithm under development. Considering individual robots as SCEs does not fit the scope of the ASCENS project, in that SC do not join or leave the system dynamically. Moreover, this approach increases considerably the complexity of system design and analysis. Thus, for the purposes of the ASCENS project, we chose to limit our scope to the first approach—considering single robots as service components. In this way, we could target the most interesting aspects of ensemble coordination directly.

A particularly important aspect for ASCENS is the fact that robot swarms possess a dual nature. Being physical objects acting in an environment, robots can be modeled through classical mechanics as bodies interacting through forces (e.g. motion, collisions, assembly, transport). At the same time, a robot swarm can be seen as a classical communication network, in which robots exchange messages to achieve coordination.

This dual nature of robot swarms affects every phase of the ensemble development life cycle. Requirement specification, for instance, might include statements regarding the correctness of the swarm state throughout an experiment. Such statement might include spatial aspects, such as moving while maintaining a cohesive formation (also known as *flocking* [30]), as well as network aspects, such as achieving consensus on the direction to follow. By the same token, modeling might need to consider the position of each robot at any time during an experiment (space), as well as the opinion of each robot on the direction to follow (network) [32].

The duality of robot swarms is apparent also in the so-called *global-to-local* problem [39,40]. The goals of a swarm, as well as its properties, are typically expressed and analyzed at the global (i.e., swarm) level. However, the actions that realize the dynamics of a swarm are executed at the local level (i.e., by each robot individually). A principled methodology to map local actions to global properties is currently an open problem, for which research is ongoing [10,16,34,19].

The design of the robotics scenario for ASCENS follows these considerations. The primary aim of the scenario was to expose the ASCENS researchers to complex, real-world problems for which partial or no solutions exist today.

2.2 The marXbot Robot

The marXbot [4] is a mobile robot developed during the Swarmanoid project [12] and the ASCENS project.[4] The marXbot is equipped with several devices that allow it to sense and act in the environment. The marXbot's modular architecture renders it easy to add new devices and configure the robot to suit the needs of particular experiments.

Lower Module. The marXbot is a non-holonomic, differential-drive robot equipped with a combination of wheels and tracks named *treels*. The treels allow the marXbot to move on mildly rough terrain while maintaining good stability. A ring of 24 equally-spaced infrared sensors placed around the lower module of the robot body double as *proximity sensors* and *light sensors*. Through these sensors, the marXbot can be programmed to avoid close obstacles and to detect the direction to a light source. The lower module also offers two sets of *ground sensors*. The first set is composed of 4 sensors located close to the treel motors, which allow the marXbot to detect 255 levels of gray on the ground. The second set, composed of 8 sensors intertwined with the infrared sensors, provides the robot with binary information to detect the presence or absence of holes on the ground.

LED-Gripper Module. Above the lower module, the marXbot houses a multipurpose module. It is designed to allow two marXbots to dock into each other

[4] The robot is also called *foot-bot* to highlight its capabilities with respect to the other Swarmanoid robots, the *hand-bot* and the *eye-bot*.

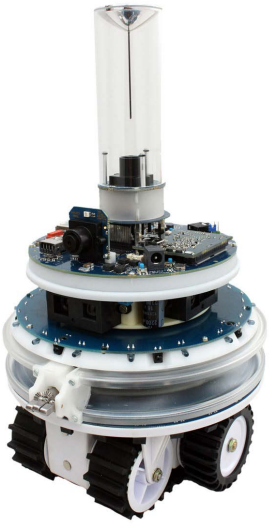


Fig. 1. The marXbot robot is a modular robot that can be configured to suit the needs of the experimenter. License: Creative Commons 3.0.

to form complex multi-robot assemblies. To this aim, the module is composed of a gripper designed to lock inside the same module of a kin robot. The gripper can be rotated freely around the yaw axis of the robot, making it possible for complex assemblies to move while connected. Another important feature of this module is the presence of 8 RGB LED embedded in the module frame. The color of each LED can be set independently and is detectable through the cameras. With these LEDs, a robot can convey its state or encode directional information for other robots.

Range-and-Bearing Module. The *range-and-bearing communication system* [33] is located above the LED-gripper module. This device allows two marXbots to exchange 12 bytes of data every 100 ms. The particularity of this device is that each robot, upon receipt of a message, also detects the location (distance and angle) of the message sender with respect to its own reference frame. This device realizes the notion of *situated communication*, an important communication modality to achieve coordination in swarm systems [35].

Distance Scanner Module. The marXbot also offers a *long-range rotating distance scanner*, which can be used to map the surroundings and to localize the robot in a static environment [25].

Top Module. The top module equips the marXbot with two cameras: *(i)* an *omni-directional camera*, whose images are analyzed to detect colored blobs

around the robot; and *(ii)* a *perspective camera*, that can be oriented frontally or towards the ceiling to detect objects. The top module also offers a *beacon*, a high-power RGB LED that can be used in combination with the cameras to highlight the position of a robot and convey its state through specific colors. Finally, the top module is also home to the Linux board of the robot, equipped with a 512 MHz ARM7 processor and 256 Mb of RAM.

2.3 General Scenario Description

The application scenario can be summarized as *disaster recovery*. We imagine that a disaster happened, such as the catastrophic failure of a nuclear plant, or a major fire in a large building. We also imagine that an activity of search-and-rescue must be performed. For instance, people may be trapped inside the building and they must be found and brought to safety. Given the high danger of operating in such environment, it is realistic to think that an ensemble of robots could be used to perform the most dangerous activities. Among these activities, two are the focus of our attention: exploring the environment and finding targets to rescue.

The screenshot in Figure 2 depicts an instantiation of the essential elements of the scenario. The environment is a large rectangular area structured by several walls. The victims to find are scattered throughout the environment. For the purposes of the ASCENS project, there was no real need to design a specific object to be retrieved. Thus, we used a marXbot that we suppose unable to move. This choice enabled us to test variants of the scenario in which the object is able to signal its location to nearby robots, and variants in which the object is completely passive. The robots are initially deployed in the *deployment area* marked in gray in Figure 2.

An important constraint is the fact that robots possess limited battery lifetime. The exhaustion of battery power is as critical an hazard as exposure to radiations. In fact, in low battery power conditions, various sensors tend to provide noisy or wrong readings, which in turn affect a robot's performance. The complete exhaustion of battery power is equivalent to the loss of a robot.

2.4 Parameters

The scenario can be formalized in a matrix of parametric activities with "tunable" complexity as illustrated in Table 1. Within ASCENS, the aim of such complexity matrix was not to enumerate the entire set of possibilities we intended to tackle—such set is too wide and general to be studied realistically. Rather, complexity tuning enabled us to isolate the relevant aspects of a certain problem, and develop new algorithms in a manageable, step-by-step process whereby further complexity was introduced gradually. In the following, we present the main features of the complexity matrix.

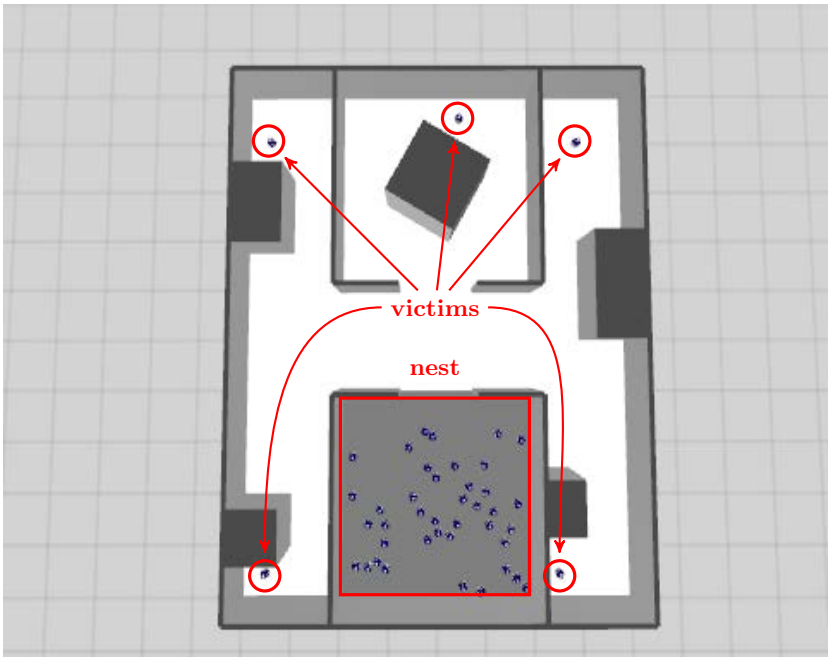


Fig. 2. The environment in which we studied collective exploration. Screenshot taken with the ARGoS robot simulator.

Exploration. To rescue the victims, the robots must first find them. Exploration serves this purpose. Exploration complexity depends on a number of factors related to the environment. Depending on the number of robots, a small environment is easier to navigate than a large one. Navigation is also easier in an obstacle-free environment than in a cluttered one. Typically, in a small, obstacle-free environment the best exploration strategy is diffusion through random walk. In a large, maze-like environment, more complex strategies are necessary. Analogously, navigation is simpler on a flat terrain than on a rough one. Another important aspect is whether the robots can exploit a map of the environment or not. The easiest situation is when a map is available beforehand. In this case, the robots can use this information to locate themselves and the interesting points in the environment, making navigation easier. Alternatively, a map could be constructed during the experiment through SLAM (simultaneous localization and mapping) techniques. The third and most challenging option is that the robots do not possess nor construct a map, but navigate in a cooperative way. An algorithm demonstrating the latter option is presented in Section 4.

Task Allocation. Task allocation is the activity of assigning robots to specific tasks [17]. In this scenario, tasks can be manifold. For instance, some robots could be explorers, other transporters. Transport, in turn, could require co-

Table 1. Complexity matrix for the robotics case study scenario

Activity	Parameter	Alternatives
Exploration	Environment size	Small/Large
	Environment structure	Obstacle-free/Cluttered
	Terrain	Flat/Rough
	Map	Available/Computable/Not available
Task allocation	Task-robot mapping	STSR/STMR
	Task dependency	Independent/Sequential/Complex
	Task assignment	Instantaneous/Time-extended
	Task dynamics	Simple/Complex
	Task distribution	Simple/Complex

operation by many robots. In general, we can distinguish between *single-* and *multi-robot tasks*, and between *single-* and *multi-task robots*. Single-robot tasks can be executed by a robot individually, while multi-robot tasks require cooperation of a group of robots. Single-task robots can execute only one task at a time, while multi-task robots can execute more than one in parallel. In our complexity matrix, we consider only the following two cases: single-task-single-robot (STSR), and single-task-multi-robot (STMR). An example of a task that can be declined in these variants is transport. STSR transport is when an object is light enough for a robot to move it. If the object requires many robots to move it, transport is STMR. Furthermore, in a realistic scenario, tasks may possess activation dynamics, i.e., each task must be executed in certain time periods [8]. We can model this by defining a function $T_i(t)$ such that its value over time t is 1 when task $i \in [1, K]$ is active, and 0 otherwise. In general, $T_i(t)$ takes the form of a square wave function, i.e., a task undergoes periods of activation and periods of de-activation. Task activation periods can be correlated to each other, for instance when some tasks are dependent on other tasks (e.g., task i must be executed before task j). Furthermore, assignment of tasks to robots can be *time-extended* or *instantaneous*. In time-extended assignment, $T_i(t)$ (or an approximation of it) is assumed known and tasks are assigned to robots according to a pre-calculated schedule. Instantaneous assignment refers to methods in which $T_i(t)$ is not known. Another important aspect in task allocation is the distribution of tasks in the environment. Task distribution has consequences on the efficiency of task discovery and execution by the robots. Task distribution is linked to the organization of the environment, i.e., how cluttered or structured the environment is. When dealing with robot swarms, in general a task must be executed by a certain number of robots, called *quota*. In practical problems, quotas are rarely precise. For example, moving a heavy object requires a minimum number of robots to compensate for the object weight. Employing more robots usually results in better performance (i.e., the object is transported faster or with less effort by the robots' motors). However, above a certain number of robots, coordination becomes an issue that negatively impacts performance. Therefore, typically quotas can be expressed as ranges [min,max].

3 The Robotics Scenario and the EDLC

The Ensemble Development Life Cycle (EDCL) introduced in Chapter III.1 [22] is composed of several phases. In this section, we report on the main findings regarding each phase. A case study relating several phases can be found in [38].

3.1 Requirement Engineering

Property-Driven Design. As explained in Section 2.1, the dynamics of a robot ensemble comprises two levels—the ensemble level and the individual level. The requirements are typically expressed at the ensemble level, but the mechanisms that realize the wanted behavior are executed at the individual level. A natural approach to reconcile the two levels is to work in step-by-step fashion, gradually refining the ensemble requirements by expressing them in more detailed forms that, eventually, lead to a practical implementation. This idea is the core of the work of Brambilla *et al.* [6], who demonstrated their approach on typical swarm behaviors such as aggregation and foraging.

Engineering Self-organization and Emergence. In Chapter III.2 [27], Noël and Zambonelli illustrate a number of methodological guidelines to engineer the basic self-organization mechanisms that lead to coordinated ensemble behaviors. The author demonstrate their approach through a variant of the scenario in which the robots must spread in an unknown environment and find victims.

3.2 Modeling/Programming and Verificaton/Validation

SCEL Modeling. In Chapter I.1 [26], De Nicola *et al.* present a complete SCEL model of a scenario variant in which robots must find and rescue victims. The robots can take the role of *explorers* or *rescuers*. Explorers search for victims; when a robot detects a victim, it becomes a rescuer. A rescuer, beside assisting a victim, informs other robots of the victim's position, thus attracting more rescuers. The SCEL model considers also the possibility that the battery charge reaches a low level, in which case the robots pause their activity and turn to the battery charging state. The authors describe two models: one based on PSCEL (a SCEL variant which includes *policies*), and one based on StocS (a stochastic extension of the SCEL semantics).

jRESP Implementation. In Chapter I.1 [26], De Nicola *et al.* also describe an implementation of the SCEL model in the jRESP framework, a Java runtime environment that realizes the SCEL paradigm. The remarkable aspect of this exercise is that the primitive concepts of jRESP closely resemble those of SCEL. Thus, through jRESP, an abstract model of a distributed algorithm for robotics can find a direct, practical implementation whose performance can be studied and characterized. In fact, jRESP programs can be simulated and analyzed through a statistical model checker. De Nicola *et al.* report the results of

such an analysis on the robotics scenario, studying the probability that a victim is rescued within a given time using different numbers of robots.

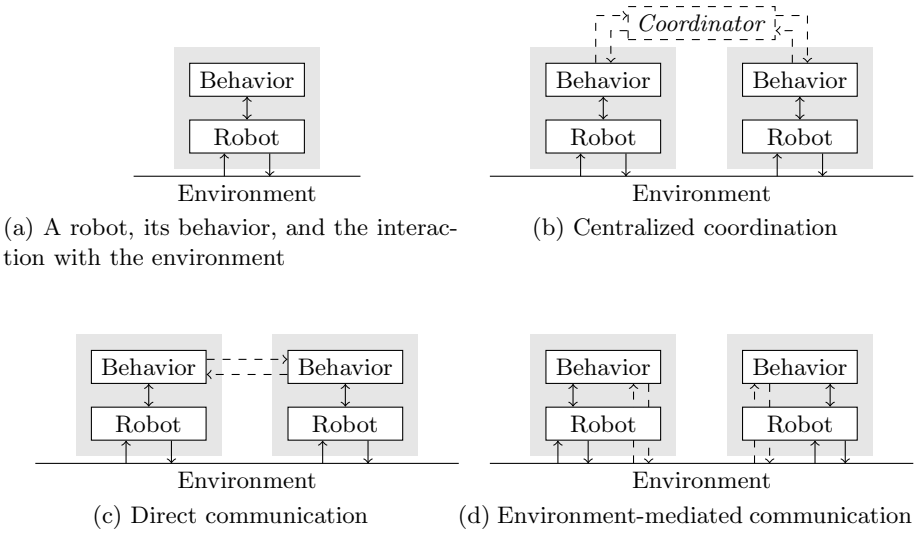
Maude Implementation. Another contribution of Chapter I.1 [26] is an analysis of a specific aspect of the scenario modeled in SCEL through a tool called MISSCEL (Maude Interpreter and Simulator for SCEL). MISSCEL is an implementation of SCEL in the Maude framework, a software for model checking. De Nicola *et al.* focus on collision avoidance, a basic behavior the robots perform while exploring the environment. In particular, they analyze the efficiency of collision avoidance when the robots are informed (i.e., can use the proximity sensors) and uninformed (i.e., they choose their direction at random).

Physics-Based Modeling and Implementation. A common technique to study behaviors in robotics is employing physics-based simulation. The advantage of this kind of simulation is the close resemblance of the simulated system dynamics with respect to its real counterpart. Physics-based simulation typically include every relevant aspect that affects the behavior of the robot ensemble—body collisions, network communication errors, etc. For the work in ASCENS, we employed the ARGoS multi-robot simulator [31], a state-of-the-art software capable of accurately simulating experiments involving thousands of robots in a fraction of real time. An example experiment developed with ARGoS is presented in Section 4.

SMC-BIP Verification. In Chapter I.3 [9], Combaz *et al.* present an approach to the verification of distributed robot behaviors based on the BIP statistical model checker. The main advantage of BIP over other modeling techniques is that BIP models can be transformed into executable programs *automatically*, making it possible to link modeling and implementation seamlessly. The authors model the scenario variant described in detail in Section 4, analyzing the effects of several alternatives for each robot behavior on the overall system performance.

3.3 Awareness and Adaptation

The notion of *awareness* and *adaptation* in robot swarms can manifest themselves at the individual level and at the ensemble level. For the purposes of ASCENS, our primary focus is modeling and achieving ensemble-level awareness and adaptation. However, the two levels are deeply intertwined—a study of ensemble awareness/adaptation cannot neglect the individual level. *Individual* awareness and adaptation can be defined as the ability of the robot to estimate its own state, as well as a relevant portion of the ensemble state, and react effectively to state changes. By *relevant portion*, here we mean that the robot must be capable of retrieving enough information about the ensemble state to make decisions leading to correct ensemble behaviors. *Ensemble* awareness and adaptation refer to the capability of the ensemble to behave as a coherent unit, by distributing information correctly and acting in a coordinated fashion.



(a) A robot, its behavior, and the interaction with the environment

(b) Centralized coordination

(c) Direct communication

(d) Environment-mediated communication

Fig. 3. Coordination patterns for groups of robots. The solid lines indicate generic interactions among entities. The dashed lines indicate coordination-aimed interactions among entities.

The relationship between the individual and the ensemble levels is complex. For instance, a high degree of individual awareness is not required to produce complex ensemble behaviors which display high degrees of awareness [24]. Research on social insects show that individuals following simple rules based on short-range information about the environment are capable of highly complex and efficient behaviors such as nest construction and food foraging. The algorithm described in Section 4 is an example of an individual behavior based on short-range information and little individual awareness that result in a complex ensemble behavior.

Adaptation Patterns. In the robotics case study, each individual robot is considered as a Service Component (SC). Each SC is associated to a program that controls its actions, here referred to as *behavior* (see Figure 3a). Groups of connected robots (physically or networked) form Service Component Ensembles. To achieve adaptation in robot ensembles, we identify four general patterns. These adaptation patterns can be expressed following the approach described in Chapter III.1 [22] for the mapping between SCs and autonomic managers. In this context, the robots are *proactive service components*, and the concept of robot behavior coincides with that of *internal autonomic manager*. The adaptation patterns can be classified into two general categories: *(i)* patterns that include an element of centralization, and *(ii)* fully distributed patterns. In patterns that include an element of centralization, such element is typically meant as dedicated SCs that collect information from the robot SCE, make decisions, and instruct

the robots accordingly (see Figure 3b). In the approach of Chapter III.1 [22] this SC is an *external autonomic manager*. In fully distributed adaptation patterns, the main coordination means is inter-robot communication. Communication can occur in two ways: either directly (a robot explicitly sends a message to another robot, Figure 3c), or indirectly (a robot reacts to the changes in the environment made by other robots, Figure 3d). Indirect, or environment-mediated communication, is also known as *stigmergy* [18].

Black-Box and White-Box Adaptation. In Chapter II.1 [7], Bruni *et al.* employ the robotics scenario depicted in Figure 2 as a testbed to validate a unified approach to both *black-box* adaptation (i.e., adaptation behaviors as they appear to an outside viewer) and *white-box* adaptation (i.e., adaptation mechanisms that affect the internal behavior of the system).

Reasoning and Learning for Awareness and Adaptation. In Chapter II.4 [21], Hölzl *et al.* propose a modeling approach called Extended Behavor Trees (XBTs). This approach targets hierarchical, concurrent behaviors that interleave reasoning, learning, and actions. XBTs can be translated into SCEL, thus integrating the EDLC and enriching its scope. The approach is validated on a variant of the proposed scenario.

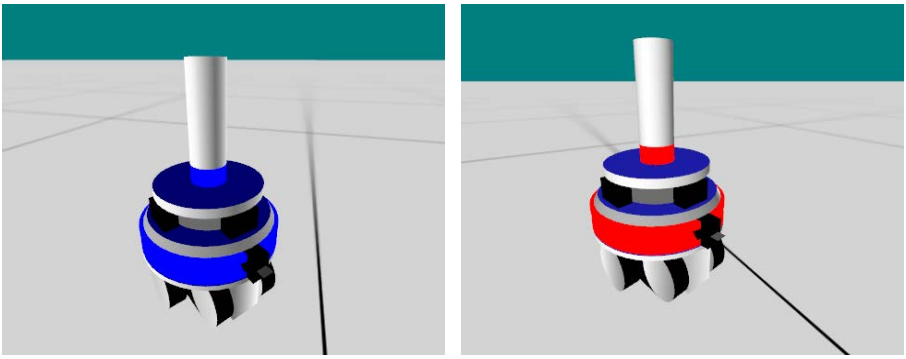
4 Implementation and Demonstration

In this section, we present a fully distributed algorithm for collective exploration. The algorithm works under the assumption that the robots are initially unaware of the whereabouts of the victims and of the structure of the environment. The concepts of awareness and adaptation play a fundamental role in this application.

In terms of awareness, as discussed in Section 3.3, the most important requirement is that the ensemble *as a whole* is capable of representing the current knowledge regarding the structure of the environment. The ultimate purpose of exploration is to allow a second set of robots, the *rescuers*, to reach the victims that need assistance.

To achieve this result, one could endow each robot with an algorithm for simultaneous localization and mapping (SLAM) [37] and let the robots integrate each others' maps through communication. With this approach, the representation of the whole environment is a composition of the individual representations of each robot. While this approach is effective, it requires adequate sensing and computation capabilities on the robots, which are mostly lacking on the marXbot. Moreover, this approach does not target the intrinsically *distributed* nature of the systems we studied throughout the project—in principle, a robot could solve the exploration task alone, given sufficient time and resources.

In this section we focus on an alternative solution, in which the robots construct a coherent collective representation of the environment without requiring



(a) An explorer robot (b) Victims are simulated with robots

Fig. 4. The robots involved in the exploration scenario. Screenshot taken with the ARGoS robot simulator.

SLAM capabilities. In terms of awareness, this algorithm demonstrates how little (or even zero) individual awareness can result in effective and coherent ensemble awareness.

4.1 Scenario Instantiation

The scenario consists of a structured environment of width W and depth D, initially unknown to the robots. As reported in Figure 2, the structure of the environment mimics that of a building floor. A team of R robots called *explorers* (Figure 4a) is deployed in a special area called the *nest* within the environment. The size of the nest is always assumed sufficient to house the entire explorer ensemble.

We imagine that a number V of victims (Fig. 4b) are scattered throughout the environment and must be found by the robots. The robots construct a representation of the environment such that a second robot ensemble, the *rescuers*, can promptly reach the victims.

4.2 Algorithm Structure

The core idea behind the algorithm is to employ the robots as *landmarks*. A landmark robot occupies a specific location of the environment and maintains communication with a number of immediate neighboring landmarks. Upon receipt of a request for direction to a specific victim by a wandering robot, two situations can occur:

1. The landmark can see the victim directly: in this case, the landmark sends the direction to the victim;
2. The landmark cannot see the victim: in this case, the landmark propagates the request to its neighbors, and then picks the shortest suggested path.

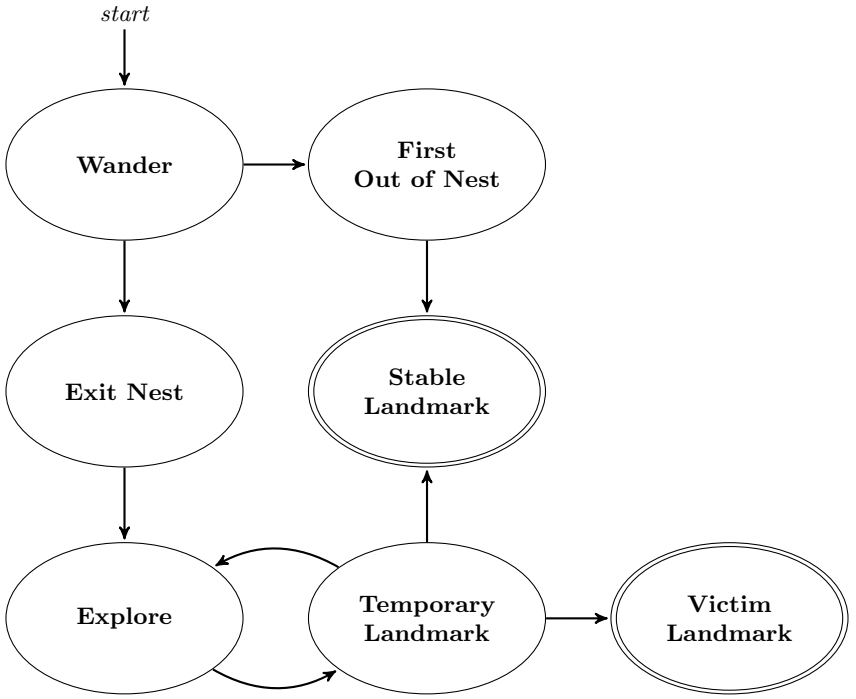


Fig. 5. A finite state machine representation of the exploration algorithm. Double-bordered nodes represent final behaviors, i.e., behaviors after which no further transition is possible.

The algorithm presented here concentrates on the creation of the network of landmarks and is inspired to the approach of Nouyan *et al.* [29]. For an algorithm that uses the landmark network to guide robots to their destination, see Ducatelle *et al.* [14].

A diagrammatic representation of the algorithm is reported in Fig. 5, while the main phases of a typical execution of this behavior are illustrated in Figure 6. In the rest of this section, we will present the main behaviors along with a snippet of their implementation in the Lua language.

Wander. The robots are initially deployed in the nest. Their first task is to find the exit of this area, which leads to the environment to explore. This first behavior makes the robot navigate randomly following an adapted version of the diffusion algorithm of Howard *et al.* [23]. To facilitate the detection of the nest exit, we color-coded the ground. The nest ground is gray, while the rest of the environment is white. Through its ground sensors, a marXbot can monitor the floor color, thus detecting when it exits the nest.

```
function rescuer:wander()
  -- State transition logic
  if rescuer:should_exit() then
    -- The robot should exit the nest because there is
    -- a landmark nearby, or because other robots are
    -- already exiting
    rescuer:switch_to_exiting()
    return
  end
  -- If we get here, the robot is out of the nest, nobody
  -- else is exiting, and no landmark is nearby
  if rescuer:is_out_of_nest() then
    -- The robot just exited the nest
    -- It's the first, so become a landmark
    rescuer:set_state(RESCUER_STATE__FIRST_LANDMARK,
                rescuer.first_landmark)
    return
  end
  -- State logic
  -- Get vector to escape from obstacles
  local repulsion = rescuer:repulsion_vector()
  if(repulsion.x * repulsion.x +
     repulsion.y*repulsion.y > 0.001) then
    rescuer:vector_to_wheel_velocity_noscale(repulsion)
  else
    robot.wheels.set_velocity(5,5)
  end
end
```

First Out of Nest. A robot switches to this behavior when its ground sensors detect white and no robot in range is in this behavior nor in any landmark-related behaviors. When a robot is in this behavior, it keeps moving for a few seconds to free space in front of the nest exit. Subsequently, the robot switches to **Stable Landmark**. It is not strictly necessary to ensure that a single robot is the 'first out of nest'. The probability that more than one robot follow this behavior is related to the ease with which a robot can find the exit of the nest (e.g., the width of the exit, the initial position and the sensor range of the robot).

```
function rescuer:first_landmark()
  -- State transition logic
  rescuer.counter = rescuer.counter + 1
  -- If 15 seconds have expired, become landmark
  if rescuer.counter > 150 then
    -- Become a stable landmark
    rescuer:set_state(RESCUER_STATE__STABLE_LANDMARK,
                      rescuer.stable_landmark)
    rescuer.landmark_data.mark = 1
    -- Stop the robot
    robot.wheels.set_velocity(0,0)
```

```
    -- Change LED color to green, for visual confirmation
    robot.leds.set_all_colors("green")
    -- Set the mark for the current landmark
    robot.range_and_bearing.
      set_data(3, rescuer.landmark_data.mark)
    robot.debug.message = robot.debug.message .. "(1)"
    return
  end
  -- State logic
  -- Just keep going straight
  robot.wheels.set_velocity(5,5)
end
```

Stable Landmark. A stable landmark is a robot that occupies a specific location of the environment and acts as a node in the communication network. A stable landmark receives requests for direction, propagates them to neighbors, and returns an answer to the robot which issued the request. For the purposes of this algorithm, once a robot has become a stable landmark, it simply acts as a beacon signalling its own position.

Exit Nest. The robots that are following the **Wandering** behavior close to the nest exit detect when the first stable landmark appears. Upon detecting this event, a robot switches to the **Exit Nest** behavior. In this behavior, the robot propagates the information about the direction to the exit throughout its neighbors. In this way, the robots that cannot detect the first landmark directly are informed of its presence and switch to this behavior as well. To exit the nest, a robot follows the direction to the landmark, if directly visible, or to the closest robot that is aware of such direction. When a robot exits the nest, it switches to the **Explore** behavior.

```
function rescuer:exit_nest()
  -- Buffer for the averaged sum of contributions of exiting
  -- robots nearby
  local exiting = { count = 0, accum = {x = 0, y = 0} }
  -- Buffer for the direction to the closest landmark
  local landmark = {
    direct = false,
    dist = INF_DISTANCE,
    angle = 0 }
  -- The current RAB message being processed
  local msg
  -- Go through RAB messages
  for i=1,#robot.range_and_bearing do
    msg = robot.range_and_bearing[i]
    if (msg.data[2] >= RESCUER_STATE__TEMPORARY_LANDMARK) and
      (msg.range < landmark.dist) then
      -- Landmark detected, and it's the closest so far
```

```
    landmark.dist = msg.range
    landmark.angle = msg.horizontal_bearing
    landmark.direct = true
  elseif msg.data[2] == RESCUER_STATE__EXIT_NEST then
    -- Exiting robot detected
    exiting.count = exiting.count + 1
    -- Calculate distance to landmark of this robot
    local land_dist =
      msg.data[4] + msg.data[3] * 256 + msg.range
    if land_dist < landmark.dist then
      -- Found the robot who knows the closest way to a
      -- landmark
      landmark.dist = land_dist
      landmark.angle = msg.horizontal_bearing
      landmark.direct = false
    end
    -- Calculate the contribution of this robot
    local lj = rescuer:lennard_jones(
      msg.range,
      RESCUER_EXITING_DISTANCE,
      RESCUER_EXITING_GAIN)
    local contr = {
      x = lj * math.cos(msg.horizontal_bearing),
      y = lj * math.sin(msg.horizontal_bearing)
    }
    exiting.accum.x = exiting.accum.x + contr.x
    exiting.accum.y = exiting.accum.y + contr.y
  end
end
-- State transition logic
-- If you can see the landmark directly and you're out
-- of the nest, explore
if landmark.direct and rescuer:is_out_of_nest() then
  rescuer:switch_to_explore()
  return
end
-- State logic
-- Postprocess the data collected
-- Take the average of the exiting robot interaction
if(exiting.count > 1) then
  exiting.accum.x = exiting.accum.x / exiting.count
  exiting.accum.y = exiting.accum.y / exiting.count
end
-- Calculate the LJ interaction to the landmark
local landmark_contr = { x = 0, y = 0 }
if landmark.dist < INF_DISTANCE then
  magnitude = 2
  if landmark.dist < 1.5 * RAB_RANGE then
    magnitude = rescuer:lennard_jones(
      landmark.dist,
```

```
        RESCUER_LANDMARK_DISTANCE,
        RESCUER_LANDMARK_GAIN)
    end
    landmark_contr.x = magnitude * math.cos(landmark.angle)
    landmark_contr.y = magnitude * math.sin(landmark.angle)
    -- Send around the closest direction to landmark known
    robot.range_and_bearing.set_data(3, landmark.dist / 256)
    robot.range_and_bearing.set_data(4, landmark.dist % 256)
  else
    robot.range_and_bearing.set_data(3, 256)
    robot.range_and_bearing.set_data(4, 256)
  end
  -- Calculate the direction
  local direction = {
    x = exiting.accum.x + landmark_contr.x,
    y = exiting.accum.y + landmark_contr.y
  }
  -- Actuate wheels
  rescuer:vector_to_wheel_velocity_scale(direction)
end
```

Explore. A robot in this behavior performs random walk in the environment. While wandering, the robot keeps track of the closest landmark detected. If the distance to this landmark becomes too high (i.e., more than 80% of the maximum range of the range-and-bearing system), the exploring robot stops and becomes a **Temporary Landmark.**

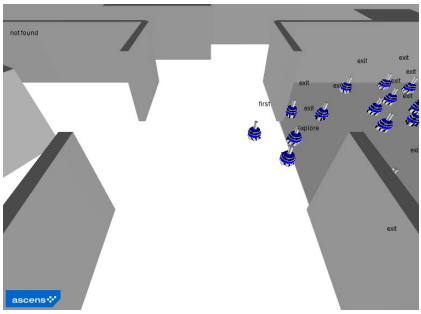
```
function rescuer:explore()
  -- State transition logic
  if rescuer:is_out_of_nest() then
    -- Get the landmarks around
    local landmarks = rescuer:landmarks_in_range()
    if landmarks then
      -- Get the data of the closest landmark
      local dist = RAB_RANGE
      local marker
      local is_victim_landmark = false
      for i = 1, #landmarks do
        if landmarks[i].range < dist then
          dist = landmarks[i].range
          marker = landmarks[i].data[3]
          is_victim_landmark =
            (landmarks[i].data[2] ==
             RESCUER_STATE__VICTIM_LANDMARK)
        end
      end
      -- Are we getting too far from the closest?
      if (not is_victim_landmark) and
        (dist > 0.8 * RAB_RANGE) then
```

```
        -- The closest landmark is getting too far
        -- Become a landmark!
        rescuer:become_landmark(marker)
        return
      end
    end
  else
    -- Explorer got back to the nest
    -- Switch back to exiting state
    rescuer:switch_to_exiting()
    return
  end
  -- State logic
  -- Wander in the environment
  local repulsion = rescuer:repulsion_vector()
  if(repulsion.x * repulsion.x +
     repulsion.y * repulsion.y > 0.001) then
    rescuer:vector_to_wheel_velocity_noscale(repulsion)
  else
    robot.wheels.set_velocity(5,5)
  end
end
```

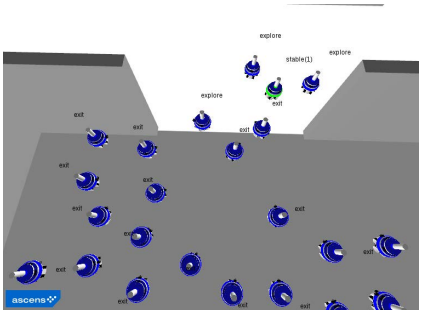
Temporary Landmark. When a robot switches to this behavior, it stops its motion and waits for a few seconds while monitoring the environment for other nearby landmarks. If a nearby landmark is located and is too close, the robot switches back **Explore**. Otherwise, at the end of the monitoring period, the robot switches to **Stable Landmark** or **Victim Landmark**, depending on whether a victim is visible or not. The rationale for this behavior is to optimize the diffusion of landmarks across the environment. The motion of explorers around a temporary landmark might hide (for a short period) the presence of other stable landmarks; the monitoring period is designed to allow the robot to collect information and discover nearby landmarks despite the motion of the explorers.

```
function rescuer:temporary_landmark()
  -- Increase counter
  rescuer.counter = rescuer.counter + 1
  -- Switch green LEDs depending on how far we are from
  -- making a decision
  if (rescuer.counter %
      RESCUER_TEMPORARY_PROGRESS_PERIOD) == 0 then
    robot.leds.set_single_color(
      rescuer.counter / RESCUER_TEMPORARY_PROGRESS_PERIOD,
      "green")
  end
  -- Collect data
  -- Go through the messages
  if #robot.range_and_bearing > 0 then
```

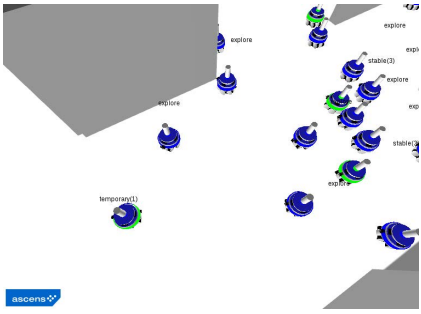
```
  -- local msg
  for i = 1, #robot.range_and_bearing do
    local msg = robot.range_and_bearing[i]
    if msg.data[1] == ROLE__VICTIM then
      -- Detected a victim in range
      rescuer.landmark_data.victim_nearby = true
    elseif msg.data[2] >=
             RESCUER_STATE__TEMPORARY_LANDMARK then
      -- Detected a landmark in range
      if rescuer.landmark_data.dist_to_closest_landmark >
           msg.range then
        rescuer.landmark_data.
          dist_to_closest_landmark = msg.range
      end
      if msg.data[2] == RESCUER_STATE__VICTIM_LANDMARK then
        rescuer.landmark_data.victim_landmark_nearby = true
      end
    end
  end
end
-- If 10 seconds have expired, make a decision
if rescuer.counter > RESCUER_TEMPORARY_PERIOD then
  if rescuer.landmark_data.victim_nearby and
    (not rescuer.landmark_data.victim_landmark_nearby) then
    -- There's a victim and no victim landmark
    -- Become victim landmark
    rescuer:set_state(RESCUER_STATE__VICTIM_LANDMARK,
                      rescuer.victim_landmark)
    robot.debug.message =
      robot.debug.message .. "(" ..
      rescuer.landmark_data.mark .. ")"
    -- Set the mark for the current landmark
    robot.range_and_bearing.
      set_data(3, rescuer.landmark_data.mark)
  elseif (not rescuer.landmark_data.victim_nearby) and
       (rescuer.landmark_data.dist_to_closest_landmark >
          0.3 * RAB_RANGE) then
    -- No victim around and no landmark is too close
    -- Become a stable landmark
    rescuer:set_state(RESCUER_STATE__STABLE_LANDMARK,
                      rescuer.stable_landmark)
    robot.debug.message =
      robot.debug.message .. "(" ..
      rescuer.landmark_data.mark .. ")"
    -- Set the mark for the current landmark
    robot.range_and_bearing.
      set_data(3, rescuer.landmark_data.mark)
  else
    -- Either there's both a victim nearby and a victim
    -- landmark, or there's no victim but a landmark is too
    -- close. Either case, go back exploring
```



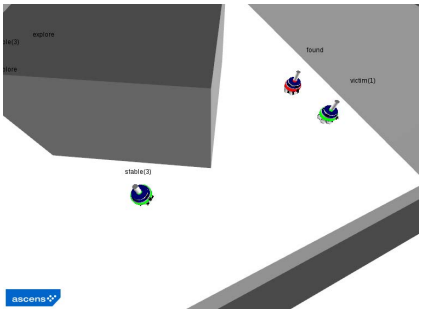
(a) The first explorer exits the nest and becomes a stable landmark.



(b) The other robots exit the nest.



(c) The explorers navigate the environment, occasionally becoming stable landmarks.



(d) Explorers that are close to a victim become victim landmarks.

Fig. 6. The essential phases of the exploration behavior. Screenshots taken with the ARGoS robot simulator

```
      rescuer:switch_to_explore()
    end
  end
end
```

Victim Landmark. When a robot is eligible to become a stable landmark, it checks for the presence of nearby victims. If at least a victim is detected, the robot becomes a victim landmark. This behavior is similar to a stable landmark in that a robot becomes part of the communication network, receiving and replying requests from the rescuers. However, the role of a victim landmark is to act as the leaf node of the network when the direction to a victim in range is requested. For the purposes of this algorithm, once a robot has become a victim landmark, it simply acts as a beacon signalling its own position.

5 Conclusions

In this chapter, we presented the robotics scenario used throughout the ASCENS project. The scenario imagines that a disaster happened in an area whose structure is unknown. Victims are assumed scattered at unknown locations. A robot ensemble is deployed to the area and must save the victims.

We decoupled the scenario in a number of parametric phases, allowing the ASCENS researchers to "tune" the complexity of the desired aspects at will.

The choice of this scenario stemmed from the need to expose ASCENS researchers to real-world coordination problems for robot ensembles. These problems proved useful to foster several studies spanning modeling, design, requirement specification, verification, adaptation, and awareness.

We presented an implementation that demonstrates a possible, albeit simple, solution for the scenario. This implementation has been used throughout the project as a reference, allowing researchers to analyze its properties and improve on its limitations.

References

1. Arkin, R.C.: Behavior-Based Robotics. MIT Press, Cambridge (1998)
2. Beni, G.: From Swarm Intelligence to Swarm Robotics. Swarm Robotics 3342, 1–9 (2005)
3. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Santa Fe Institute Studies in the Sciences of Complexity. Oxford University Press, New York (1999)
4. Bonani, M., Longchamp, V., Magnenat, S., Rétornaz, P., Burnier, D., Roulet, G., Vaussard, F., Bleuler, H., Mondada, F.: The marXbot, a miniature mobile robot opening new perspectives for the collective-robotic research. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4187–4193. IEEE Press, Piscataway (2010)
5. Brambilla, M., Ferrante, E., Birattari, M., Dorigo, M.: Swarm robotics: a review from the swarm engineering perspective. Swarm Intelligence 7(1), 1–41 (2013)
6. Brambilla, M., Pinciroli, C., Birattari, M., Dorigo, M.: Property-driven design for swarm robotics. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, pp. 139–146. International Foundation for Autonomous Agents and Multiagent Systems (2012)
7. Bruni, R., Corradini, A., Gadducci, F., Hölzl, M., Lafuente, A.L., Vandin, A., Wirsing, M.: Reconciling White-Box and Black-Box Perspectives on Behavioral Self-adaptation. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems. LNCS, vol. 8998, pp. 163–184. Springer, Heidelberg (2015)
8. Brutschy, A., Pini, G., Pinciroli, C., Birattari, M., Dorigo, M.: Self-organized task allocation to sequentially interdependent tasks in swarm robotics. Autonomous Agents and Multi-Agent Systems 28(1), 101–125 (2014)
9. Combaz, J., Bensalem, S., Tiezzi, F., Margheri, A., Pugliese, R., Kofron, J.: Correctness of Service Components and Service Component Ensembles. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems. LNCS, vol. 8998, pp. 107–159. Springer, Heidelberg (2015)

10. Crespi, V., Galstyan, A., Lerman, K.: Top-down vs bottom-up methodologies in multi-agent system design. Autonomous Robots 24(3), 303–313 (2008)
11. Dorigo, M., Birattari, M., Brambilla, M.: Swarm robotics. Scholarpedia 9(1), 1463 (2014)
12. Dorigo, M., Floreano, D., Gambardella, L., Mondada, F., Nolfi, S., Baaboura, T., Birattari, M., Bonani, M., Brambilla, M., Brutschy, A., Burnier, D., Campo, A., Christensen, A., Decugnière, A., Di Caro, G., Ducatelle, F., Ferrante, E., Förster, A., Guzzi, J., Longchamp, V., Magnenat, S., Martinez Gonzales, J., Mathews, N., Montes de Oca, M., O'Grady, R., Pinciroli, C., Pini, G., Rétornaz, P., Roberts, J., Sperati, V., Stirling, T., Stranieri, A., Stützle, T., Trianni, V., Tuci, E., Turgut, A., Vaussard, F.: Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. IEEE Robotics & Automation Magazine 20(4), 60–71 (2013)
13. Dorigo, M., Birattari, M.: Swarm intelligence. Scholarpedia 2(9), 1462 (2007)
14. Ducatelle, F., Di Caro, G., Förster, A., Bonani, M., Dorigo, M., Magnenat, S., Mondada, F., O'Grady, R., Pinciroli, C., Rétornaz, P., Trianni, V., Gambardella, L.M.: Cooperative navigation in robotic swarms. Swarm Intelligence 8(1), 1–33 (2014)
15. Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., Birattari, M.: AutoMoDe: A novel approach to the automatic design of control software for robot swarms. Swarm Intelligence, 1–24 (2014)
16. Gazi, V., Fidan, B.: Coordination and control of multi-agent dynamic systems: Models and approaches. In: Şahin, E., Spears, W.M., Winfield, A.F.T. (eds.) SAB 2006. LNCS, vol. 4433, pp. 71–102. Springer, Heidelberg (2007)
17. Gerkey, B.P., Matarić, M.J.: A formal analysis and taxonomy of task allocation in multi-robot systems. The International Journal of Robotics Research 23(9), 939–954 (2004)
18. Grassé, P.: La reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. la théorie de la stigmergie: Essai d'interprétation des termites constructeurs. Insects Sociaux 6, 41–83 (1959)
19. Hamann, H.: Towards swarm calculus: Urn models of collective decisions and universal properties of swarm performance. Swarm Intelligence 7(2-3), 145–172 (2013)
20. Hinchey, M.G., Sterritt, R., Rouff, C.: Swarms and swarm intelligence. Computer 40(4), 111–113 (2007)
21. Hölzl, M., Gabor, T.: Reasoning and Learning for Awareness and Adaptation. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems. LNCS, vol. 8998, pp. 249–290. Springer, Heidelberg (2015)
22. Hölzl, M., Koch, N., Puviani, M., Wirsing, M., Zambonelli, F.: The Ensemble Development Life Cycle and Best Practices for Collective Autonomic Systems. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems. LNCS, vol. 8998, pp. 325–354. Springer, Heidelberg (2015)
23. Howard, A., Matarić, M., Sukhatme, G.: Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In: Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (DARS), pp. 299–308. Springer, New York (2002)
24. Self-organized, M.G.J.C.W.L.T.J.D.R.G.: aggregation without computation. International Journal of Robotics Research 33(8), 1145–1161 (2014)

25. Magnenat, S., Longchamp, V., Bonani, M., Rétornaz, P., Germano, P., Bleuler, H., Mondada, F.: Affordable slam through the co-design of hardware and methodology. In: 2010 IEEE International Conference on Robotics and Automation (ICRA 2010), pp. 5395–5401. IEEE Press, Piscataway (2010)
26. De Nicola, R., Latella, D., Lafuente, A.L., Loreti, M., Margheri, A., Massink, M., Morichetta, A., Pugliese, R., Tiezzi, F., Vandin, A.: The SCEL Language: Design, Implementation, Verification. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems. LNCS, vol. 8998, pp. 3–71. Springer, Heidelberg (2015)
27. Noël, V., Zambonelli, F.: Methodological Guidelines for Engineering Self-organization and Emergence. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems. LNCS, vol. 8998, pp. 355–378. Springer, Heidelberg (2015)
28. Nolfi, S., Floreano, D.: Evolutionary robotics. MIT Press, Cambridge (2000)
29. Nouyan, S., Campo, A., Dorigo, M.: Path formation in a robot swarm. Swarm Intelligence 2(1), 1–23 (2008)
30. Olfati-Saber, R.: Flocking for multi-agent dynamic systems: Algorithms and theory. IEEE Transactions on Automatic Control 51(3), 401–420 (2006)
31. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Birattari, M., Gambardella, L.M., Dorigo, M.: ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. Swarm Intelligence 6(4), 271–295 (2012)
32. Ren, W., Beard, R.: Distributed consensus in multi-vehicle cooperative control: theory and applications. Springer, Berlin (2007)
33. Roberts, J., Stirling, T., Zufferey, J.C., Floreano, D.: 2.5d infrared range and bearing system for collective robotics. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009), IEEE Press, Piscataway (2009)
34. Schmickl, T.: How to engineer robotic organisms and swarms? In: Bio-Inspired Self-Organizing Robotic Systems, pp. 25–52. Springer, Berlin (2011)
35. Støy, K.: Using situated communication in distributed autonomous mobile robots. In: Proceedings of the 7th Scandinavian Conference on Artificial Intelligence, pp. 44–52. IOS Press, Amsterdam (2001)
36. Sutton, R.S., Barto, A.G.: Introduction to reinforcement learning. MIT Press, Cambridge (1998)
37. Thrun, S., Leonard, J.J.: Simultaneous localization and mapping. In: Springer handbook of robotics, pp. 871–889. Springer, Heidelberg (2008)
38. Wirsing, M., Hölzl, M., Tribastone, M., Zambonelli, F.: ASCENS: Engineering Autonomic Service-Component Ensembles. In: Beckert, B., Damiani, F., de Boer, F.S., Bonsangue, M.M. (eds.) FMCO 2011. LNCS, vol. 7542, pp. 1–24. Springer, Heidelberg (2013), `http://www.pst.ifi.lmu.de/~hoelzl/fmco-2011.pdf`
39. Yamins, D.: Towards a theory of local to global in distributed multi-agent systems (i). In: Proceedings of the fourth international joint conference on autonomous agents and multiagent systems (AAMAS'04), pp. 183–190. ACM Press, New York (2005)
40. Yamins, D.: Towards a theory of local to global in distributed multi-agent systems (ii). In: Proceedings of the fourth international joint conference on autonomous agents and multiagent systems (AAMAS'04), pp. 191–198. ACM Press, New York (2005)