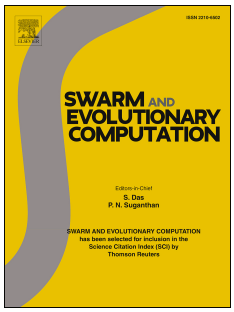
Accepted Manuscript



Make robots Be Bats: Specializing robotic swarms to the Bat algorithm

Patricia Suárez, Andrés Iglesias, Akemi Gálvez

PII: S2210-6502(17)30633-8

DOI: 10.1016/j.swevo.2018.01.005

Reference: SWEVO 346

To appear in: *Swarm and Evolutionary Computation BASE DATA*

Received Date: 24 July 2017

Revised Date: 20 November 2017

Accepted Date: 9 January 2018

Please cite this article as: P. Suárez, André. Iglesias, A. Gálvez, Make robots Be Bats: Specializing robotic swarms to the Bat algorithm, *Swarm and Evolutionary Computation BASE DATA* (2018), doi: 10.1016/j.swevo.2018.01.005.

Make Robots Be Bats: Specializing Robotic Swarms to the Bat Algorithm

Patricia Suárez[1], Andrés Iglesias[1,2,†], Akemi Gálvez[1,2]

[1] *Department of Applied Mathematics and Computational Sciences*
E.T.S.I. Caminos, Canales y Puertos, University of Cantabria
Avda. de los Castros, s/n, 39005, Santander, SPAIN
[2] *Department of Information Science, Faculty of Sciences*
Toho University, 2-2-1 Miyama
274-8510, Funabashi, JAPAN
[†] *Corresponding author: iglesias@unican.es*
http://personales.unican.es/iglesias

Abstract

Bat algorithm is a powerful nature-inspired swarm intelligence method proposed by Prof. Xin-She Yang in 2010, with remarkable applications in industrial and scientific domains. However, to the best of authors' knowledge, this algorithm has never been applied so far in the context of swarm robotics. With the aim to fill this gap, this paper introduces the first practical implementation of the bat algorithm in swarm robotics. Our implementation is performed at two levels: a physical level, where we design and build a real robotic prototype; and a computational level, where we develop a robotic simulation framework. A very important feature of our implementation is its high specialization: all (physical and logical) components are fully optimized to replicate the most relevant features of the real microbats and the bat algorithm as faithfully as possible. Our implementation has been tested by its application to the problem of finding a target location within unknown static indoor 3D environments. Our experimental results show that the behavioral patterns observed in the real and the simulated robotic swarms are very similar. This makes our robotic swarm implementation an ideal tool to explore the potential and limitations of the bat algorithm for real-world practical applications and their computer simulations.

Keywords: swarm computation, swarm robotics, bat algorithm, unknown target location, behavioral patterns.

1. Introduction

1.1. Swarm Intelligence

For many years, the concept of *intelligence* was generally perceived to be an exclusive attribute of highly sophisticated individuals. Not surprisingly, this was also the

approach taken in the early days of the artificial intelligence field. However, nature provides several examples of groups of animals that exhibit a more sophisticated social behavior than what would be possible through the simple aggregation of individual behavioral patterns. As a result, the social group is able to carry out complex tasks that their individual members cannot intend for. Take, for instance, the colonies of social insects (ants, termites, bees, fireflies), and other animal formations (fish schooling, bird flocking, animal herding, and so on). They come up with astonishingly complex social behaviors from the combined efforts of individuals with extremely limited intelligence. Amazingly, this complex collective behavior emerges from a small set of simple behavioral rules exploiting only low-level interactions between individuals and with the environment (stigmergy) using decentralized control and self-organization.

In this context, one of the most exciting breakthroughs in artificial intelligence during the last decades is the adoption and subsequent popularization of this collective intelligence arising from a collection of simple, unsophisticated, and generally homogeneous agents collaborating together to solve a complex problem. This field, globally known as *swarm intelligence* [5, 6], is overcoming the traditional mathematical approaches for solving optimization problems and laying the foundations for a new computational paradigm: the *swarm computation* [13, 27]. Under this new paradigm, there is no centralized intelligence controlling the swarm, taking decisions, and dictating how the swarm units should behave. Instead, *local* and (at some extent) *random* interactions between simple agents lead to the emergence of *global* sophisticated "intelligent" behaviors, unknown to the individual agents.

Nowadays, swarm intelligence is attracting increasing attention from researchers and practitioners owing to its potential applications to many problems. For instance, military and civil applications related to the control of unmanned vehicles have been described in [45, 46, 54]. Many other applications can also been found in the scientific literature; see [5, 19, 63] for several illustrative examples in different fields. The interested reader is also referred to [13, 27] for a comprehensive overview about the field of swarm intelligence, its history, main techniques, and applications.

1.2. Swarm Robotics

One of most relevant applications of swarm intelligence is robotics. Several scientific papers and research projects have shown that self-organizing swarm robots can potentially accomplish complex tasks and thus replace sophisticated and expensive robots by simple inexpensive drones, a research subfield usually referred to as *swarm robotics* [3, 14, 64]. The reader is kindly referred to [56] for some illustrative early examples and applications of swarm robotics; see also [50] for a more updated survey on recent advances in the field. As remarked by several authors [1, 6], swarm robotic systems offer several interesting advantages, such as:

- *Improved performance by parallelization*: swarm intelligence systems are very well suited for parallelization, because the swarm members can perform different actions at different locations simultaneously. This feature makes the swarm more flexible and efficient for complex tasks, as individual robots (or groups of them) can solve different parts of a complex task independently.

- *Task enablement*: groups of robots can do certain tasks that are impossible or very difficult for a single robot (e.g., collective transport of too heavy items, dynamic target tracking, cooperative environment monitoring, autonomous surveillance of large areas).

- *Scalability*: inclusion of new robots into a swarm does not require reprogramming the whole swarm. Furthermore, because interactions between robots involve only neighboring individuals, the total number of interactions within the system does not increase dramatically by adding new units.

- *Distributed sensing and action*: a swarm of simple interconnected mobile robots deployed throughout a large search space possesses greater exploratory capacity and a wider range of sensing than a sophisticated robot. This makes the swarm much more effective in several tasks: exploration and navigation (e.g., in disaster rescue missions), nanorobotics-based manufacturing, microbotics for human body diagnosis, and many others.

- *Fault tolerance*: due to the decentralized and self-organized nature of the swarm, the failure of a single unit does not affect the completion of the given task.

All these advantages motivated a great interest in swarm robotics during the last two decades. The reader is kindly referred to Section 2 for further details on this issue.

1.3. Nature-Inspired Metaheuristic Methods

Most of swarm intelligence methods are computational metaphors based on the dynamics of natural groups. A classical example is the *ant colony optimization* (ACO) method, based on the behavior of colonies of ants, which are able to carry out difficult tasks unattainable for individual ants [10, 11]. Another example is the behavior of a flock of birds when moving all together following a common tendency in their displacements, an inspiration to the *particle swarm optimization* (PSO) method [12, 25]. Other examples include popular optimization methods such as artificial bee colony, firefly algorithm, cuckoo search, and bat algorithm. These and many other examples lead to a valuable set of computational intelligence techniques known as *nature-inspired metaheuristic methods*. The reader is referred to [57] for a gentle introduction to several recent nature-inspired metaheuristic methods; see also [62, 63].

Among them, the *bat algorithm* is an increasingly popular swarm intelligence algorithm originally proposed by Prof. Xin-She Yang in 2010 [58, 60]. The algorithm is based on the peculiar behavior of microbats (see Section 3 for details). In our experience, the bat algorithm has shown to be a very effective method to solve complex multimodal nonlinear continuous optimization problems involving a large number of variables, such as data fitting through free-form parametric curves [20, 21, 23] and surfaces [22], and multi-objective problems [59]. On the other hand, the fundamental principle of the bat algorithm (namely, the echolocation through ultrasounds) along with some of its most important features and parameters (operating frequency, operating time, number of cycles per bust, accuracy range, traveling range of pulses, and

so on) can readily be reproduced with current hardware (see Section 4 for details). This makes the bat algorithm an excellent method for potential applications in swarm robotics. Furthermore, some recent papers have reported successful applications of the bat algorithm to some problems in robotics [15, 16]. However, these works are mostly focused on the problem of parameter tuning of PI-controllers for optimal positioning of robotic arms for safety and functionality, and have no direct relationship with the area of swarm robotics. Clearly, the application of bat algorithm to swarm robotics is a very promising yet unexplored field. In this paper we are aimed at filling this gap. Consequently, this is the algorithm used in this paper.

1.4. Main Contributions and Structure of the Paper

Despite of its remarkable features, to the best of our knowledge, the bat algorithm has never been applied so far in the context of swarm robotics. Aimed at filling this gap, a previous paper in [49] presented some initial ideas about the application of the bat algorithm to swarm robotics. However, that paper was based on very preliminary work and strongly affected by limitations of space. This paper is a substantial extension in several ways. The main contributions of this paper are:

1. First and utmost contribution of this work is *the first practical implementation of the bat algorithm for swarm robotics*. To the best of our knowledge, no previous application of the bat algorithm to swarm robotics has been reported so far in the literature.
2. Our implementation is performed at two levels: we introduce both *a physical robotic prototype* (described in Section 4.1) and *a computational robotic simulation framework* (described in Section 4.2).
3. A key differential factor of our approach is its *high specialization*. The general trend in swarm robotics is to create flexible drones able to support several algorithms with just minor (if any) modifications. Although this approach can be useful for comparative purposes among algorithms, it also yields biased comparisons since the design and components are not optimally chosen for each particular algorithm under analysis. Opposed to previous approaches in the field, our implementation has been carefully designed to replicate faithfully almost all features of the real bats and the bat algorithm (see Section 4.3 for details). In other words, all our (physical and logical) components are fully *optimized* to follow the bat algorithm principles as accurate as possible.
4. Our implementation has been applied to the problem of finding a target location within unknown static indoor 3D environments. To this aim, three different computational scenes have been considered. The first one has been recreated from the real-world storage room where we perform our experiments with the real robotic swarm.
5. Our experiments show that the behavioral patterns observed in both the real and the simulated robotic swarms are very similar. Except for small deviations due to the typical noise in real-world conditions, the behavior and evolution of the robotic swarms at the physical and computational levels match each other fairly accurately.

This paper is organized as follows: previous work in the field of swarm robotics is briefly reported in Section 2. In Section 3 we provide the reader with a gentle overview about the bat algorithm and its main rules and features. We also describe the algorithm through its pseudocode. In Section 4 we describe the main features of our implementation of the bat algorithm for swarm robotics in terms of hardware and software. The analogies and differences among the real bats, the bat algorithm, and our physical and computational swarm robotics implementation are also discussed in that section. Some computational and real-world experiments about finding a target location with our robotic swarm are reported in detail in Section 5. The issue of comparison of our approach with other alternative methods in the literature is discussed in Section 6. The paper closes with the main conclusions and some hints about our future work in the field.

2. Previous work

In this section, some previous work on swarm robotics is briefly reported. Our contribution spans both the physical and logical levels, so we organize our description on previous work accordingly, with Sections 2.1 and 2.2 devoted to the robotic platforms (physical level) and the robotic simulation frameworks (logical level), respectively.

2.1. Robotic Platforms

Swarm robotics has attracted much attention from the scientific community and the industry during the last decades. Pioneering research projects dating back the 80s and 90s (such as CEBOT, SWARMS or ACTRESS) made just preliminary advances, yet they set the foundations of the field. Since then, ambitious large-scale projects, such as the European-funded projects Swarm-bots[1] (2001-2005), i-Swarm[2] (2004-2008) and Swarmanoid[3] (2006-2010) and initiatives from USA universities such as Harvard, MIT, Stanford, UPenn, ASU, Texas A&M and many others, have led to a boost in swarm robotics. As a result, several approaches have been described in the literature during the last few years.

One of the first approaches in the field was the robot *Khepera* [34], developed at the EPFL (Lausanne, Switzerland) in mid 90s. It was sold to a thousand research labs worldwide, having served researchers for 10 years. Subsequently versions (such as *Khepera III* [37]) were released for the following decade along with some simulation platforms (see Section 2.2 for further details).

This work was later extended to a new robot, *Alice* [8, 9], developed by Gilles Caprari at the Autonomous Systems Lab at EPFL. From the very beginning, *Alice* designed to be autonomous, small (with just under 1 cubic inch, it featured a very amazingly small size for its time), and relatively inexpensive, making it affordable to construct and operate a large crowd of robots simultaneously (such as swarms of 90 robots working

[1] see URL: http://www.swarm-bots.org/
[2] see URL: http://www.i-swarm.org/
[3] see URL: http://www.swarmanoid.org/

cooperatively, a remarkable achievement at that time). These micro-robots were even for sale at one time, not only for researchers but also for hobbyists. They were also highly configurable, with numerous extension modules available for different research purposes. Some illustrative examples of applications of *Alice* include navigation and map building [7], teams of robots playing soccer cooperatively, and the embodiment of cockroach aggregation [17]. Other interesting applications can also be found in [9].

A larger (CD size) robot was *Kobot* [51, 52]. It was applied to analyze the self-organized flocking of a swarm of robots moving as a coherent group and avoiding obstacles as if they were a single "super-organism". The *e-Puck* was a popular cylindrical-shaped micro-robot designed to be robust and affordable enough to allow intensive classroom use [36]. It was quite popular for academic purposes, particularly in small amounts. However, its size and price were limiting factors for its massive deployment, as required for crowds in swarm robotics.

Another popular micro-robot was *Jasmine*[4], a public open-hardware development to create a simple and cost-effective micro-robotics platform [47]. This micro-robot has been widely used in swarm robotics studies, such as playing the role of a honeybee in aggregation scenarios [26]. A differential feature of *Jasmine* with respect to other swarms of robots is that *Jasmine* only supports local communication, while long distance communication is neither intended nor implemented, making it a good testbed for many problems. Other micro-robots well suited for swarm robotics are *S-bot* [35], *i-Swarm robot* [53], *SwarmBot* [29], *AMiR* (Autonomous Miniature Robot) [2], *Colias* [3] and its evolution, *Colias*-Φ [4].

Motivated by the recent trend of increasing the number of robots in the swarm, several relevant works are focusing on scalability issues and the challenges they present. Crowds of up to 100 robots (and even more) has been analyzed in projects, such as the *I-swarm* [48] and the *iRobot* swarm [32] projects, by using *R-one*[5], *iRobot*, *SwarmBot* and other micro-robotics models [30, 31, 44]. A significant step in this regard was the small *kilobot* [6], by Michael Rubenstein, at the Self Organizing Systems Research Group at Harvard University [39]. This robot can only perform three simple tasks: respond to light, measure a distance, and sense the presence of other robots. However, when combined, they can organize themselves into shapes, such as grouping into clusters based on their own color light (or that of their neighbors) or dispersing to fill a space. Kilobots are programed all at once, as a group, using infrared light. Each kilobot gets the same set of instructions as the next. With just a few lines of programming, they can execute together complex natural processes, such as synchronize their flashing lights like a swarm of fireflies, or moving together towards a light source similar to the way bacteria search for food [40, 41]. Recently, a swarm of one thousand kilobots has been reported in the literature [42]. With this huge amount, individual units are not really important; it does not even matter if one or a few robots break down, as the collective behavior of the swarm still prevails. In other words, a large robotic

[4]see URL: http://www.swarmrobot.org/

[5]see URL: http://mrsl.rice.edu/projects/r-one

[6]see URL: https://ssr.seas.harvard.edu/kilobots

swarm provides a lot of flexibility and robustness, as the collective tendency of the swarm is highly immune to individual failures or breakages [43]. It also gives room to analyze some interesting configurations; for instance, a gradient formation, where a source robot generates a gradient value that is incremented as it propagates through the swarm, giving each robot a metrics of the distance value from the source, or for localization tasks, where robots determine their position in the coordinate system by communicating with already localized robots [41, 43].

2.2. Robotic Simulation Frameworks

Swarm robotics is a very complex research area involving many different technologies including electronics, mechanics, physics, and computer hardware and software. In real-world experiments, it is almost impossible to make all these technologies work perfectly together. Therefore, it is common in the field to rely on realistic simulations and fast prototyping by software to reduce the burden in developing a swarm of physical robotic units, make improvements on preliminary designs, and speed up the experiments by focusing on the most critical aspects without wasting time and resources in technical issues or environmental conditions. Because of these reasons, several robotic simulation frameworks for robots and robotic swarms have been developed, with different user interfaces and technical features. Some have been created exclusively for experimentation on particular robotic platforms and are not available for public use. Others are publicly available and very popular. In this section, we describe some of the most popular robot simulators suitable for robotic swarms. Of course, this description is by no means exhaustive, as many other robot simulators have been developed during the last few years. A detailed discussion on this topic is beyond the scope of this paper.

One of the first popular open source simulators for multi-robot simulation (including swarm robotics) is *Player/Stage/Gazebo* [18], a freeware open-source programming framework comprised of three main components:

- *Player:* a language-independent and platform-independent network server for robot control providing a clean and simple interface to the robot's sensors and actuators over the IP network.
- *Stage*: a multi-robot simulator of a population of mobile robots, sensors and actuators in a 2D bitmapped environment. It is often used as a *Player* plugin module, providing populations of virtual devices for *Player*. As such, it has been successfully applied to 2D simulations of robots (see, for instance, [55]).
- *Gazebo*: a multi-robot simulator for outdoor environments in 3D. Originally designed as a 3D alternative to *Stage*, *Gazebo* has evolved into a powerful robot simulator supporting 3D indoor and outdoor environments and providing access to several physics and graphical engines. It also includes modules and support for a variety of sensors and robot models.

Another popular multi-robot simulator is *Webots* [33], a cross-platform commercial product to simulate real robots through realistic models of many of the most popular commercial robots. *SwarmBot3D* is a simulation tool for the S-bot robot of the

Swarmbot project [32, 35]. It was developed on top of *Vortex*, a commercial physics simulation engine, and describes both robots and worlds as XML text files. It also includes all hardware functionalities (sensors and mechanics) of a real S-bot as well as support for handling a group of robots either as independent units or in a swarm configuration. Another popular robot simulation program is *Microsoft Robotics Developer Studio* (Microsoft RDS for short), a Windows-based framework in C# for control and simulation of robotic units [24].

Some of the most popular robotic platforms described in previous section are also equipped with powerful simulation frameworks. For instance, several robot simulator programs have been developed for Khepera III and subsequent versions of this popular robot. Those programs include the *Khepera-Lisp Interface* (KHLI), *Khepera Simulator*, *YAKS*, *KiKS* (a *Khepera* simulator for *Matlab*), and the *Khepera III Toolbox* (a software toolbox for the *Khepera III* robot). Another example is *Jasmine*, with a simulation system based on *Breve*, an object-oriented programming language called *Steve* and sensor models for *Jasmine III* robot, and two robot simulators developed at EPFL for the popular *e-puck* robotic platform: the *ENKI* system, an open source 2D physics-based robot simulator in C++, and *V-REP*, a 3D robot simulator based on a distributed control architecture. Other notable examples include *Mona*, an open-source open-hardware platform developed at the University of Manchester, the *R-one* robotic initiative from Rice University providing an operating system and accompanying software to program the *R-one* robotic platform, a widely used robotic kit for educational and research purposes, *Kilobot* from Harvard University, and the recently released software code of the University of Colorado Boulder crowfunding *Droplets* project including a publicly accessible robot simulator.

3. The Bat Algorithm

The *bat algorithm* is a bio-inspired swarm intelligence algorithm originally proposed by Xin-She Yang in 2010 to solve optimization problems [58, 60]. The algorithm is based on the echolocation behavior of bats. The author focused particularly on microbats, as they use a type of sonar called *echolocation*, with varying pulse rates of emission and loudness, to detect prey, avoid obstacles, and locate their roosting crevices in the dark. In this paper we also adhere to this approach, so (unless otherwise stated) the term bats should be understood to refer to microbats henceforth. The interested reader is referred to the general paper in [61] for a comprehensive, updated review of the bat algorithm, its variants and applications.

3.1. Basic rules

The idealization of the echolocation of microbats can be summarized as follows (see [58] for details):

1. Bats use echolocation to sense distance and distinguish between food, prey and background barriers.
2. Each virtual bat flies randomly with a velocity $\mathbf{v}_i$ at position (solution) $\mathbf{x}_i$ with a fixed frequency f_{min}, varying wavelength λ and loudness A_0 to search for prey.

As it searches and finds its prey, it changes wavelength (or frequency) of their emitted pulses and adjust the rate of pulse emission r, depending on the proximity of the target.

3. It is assumed that the loudness will vary from an (initially large and positive) value A_0 to a minimum constant value A_{min}.

In order to apply the bat algorithm for optimization problems more efficiently, some additional assumptions are strongly advisable. In general, we assume that the frequency f evolves on a bounded interval $[f_{min}, f_{max}]$. This means that the wavelength λ is also bounded, because f and λ are related to each other by the fact that the product $\lambda.f$ is constant. For practical reasons, it is also convenient that the largest wavelength is chosen such that it is comparable to the size of the domain of interest (the search space, for optimization problems). For simplicity, we can assume that $f_{min} = 0$, so $f \in [0, f_{max}]$. The rate of pulse can simply be in the range $r \in [0, 1]$, where 0 means no pulses at all, and 1 means the maximum rate of pulse emission. With these idealized rules indicated above, the basic pseudo-code of the bat algorithm is shown in Algorithm 1. It is described in next paragraphs.

3.2. The algorithm

Basically, the algorithm considers an initial population of $\mathcal{P}$ individuals (bats). Each bat, representing a potential solution of the optimization problem, has a location $\mathbf{x}_i$ and velocity $\mathbf{v}_i$. The algorithm initializes these variables (lines 1-2) with random values within the search space. Then, the pulse frequency, pulse rate, and loudness are computed for each individual bat (lines 3-4). Then, the swarm evolves in a discrete way over generations (line 5), like time instances (line 19) until the maximum number of generations, $\mathcal{G}_{max}$, is reached (line 20). For each generation g and each bat (line 6), new frequency, location and velocity are computed (lines 7-8) according to the following evolution equations:

$$f_i^g = f_{min}^g + \beta(f_{max}^g - f_{min}^g) \quad (1)$$

$$\mathbf{v}_i^g = \mathbf{v}_i^{g-1} + [\mathbf{x}_i^{g-1} - \mathbf{x}^*]\, f_i^g \quad (2)$$

$$\mathbf{x}_i^g = \mathbf{x}_i^{g-1} + \mathbf{v}_i^g \quad (3)$$

where $\beta \in [0, 1]$ follows the random uniform distribution, and $\mathbf{x}^*$ represents the current global best location (solution), which is obtained through evaluation of the objective function at all bats and ranking of their fitness values. The superscript $(.)^g$ is used to denote the current generation g.

The best current solution and a local solution around it are probabilistically selected according to some given criteria (lines 9-12). Then, search is intensified by a local random walk (line 13). For this local search, once a solution is selected among the current best solutions, it is perturbed locally through a random walk of the form:

$$\mathbf{x}_{new} = \mathbf{x}_{old} + \epsilon \mathcal{A}^g \quad (4)$$

Require: (Initial Parameters)
Population size: $\mathcal{P}$
Maximum number of generations: $\mathcal{G}_{max}$
Loudness: $\mathcal{A}$
Pulse rate: r
Maximum frequency: f_{max}
Dimension of the problem: d
Objective function: $\phi(\mathbf{x})$, with $\mathbf{x} = (x_1, \ldots, x_d)^T$
Random number: $\theta \in U(0, 1)$

1: $g \leftarrow 0$
2: Initialize the bat population $\mathbf{x}_i$ and $\mathbf{v}_i$, $(i = 1, \ldots, n)$
3: Define pulse frequency f_i at $\mathbf{x}_i$
4: Initialize pulse rates r_i and loudness $\mathcal{A}_i$
5: **while** $g < \mathcal{G}_{max}$ **do**
6: **for** $i = 1$ **to** $\mathcal{P}$ **do**
7: Generate new solutions by adjusting frequency,
8: and updating velocities and locations //eqns. (1)-(3)
9: **if** $\theta > r_i$ **then**
10: $\mathbf{s}^{best} \leftarrow \mathbf{s}^g$ //select the best current solution
11: $\mathbf{ls}^{best} \leftarrow \mathbf{ls}^g$ //generate a local solution around $\mathbf{s}^{best}$
12: **end if**
13: Generate a new solution by local random walk
14: **if** $\theta < \mathcal{A}_i$ *and* $\phi(\mathbf{x_i}) < \phi(\mathbf{x}^*)$ **then**
15: Accept new solutions
16: Increase r_i and decrease $\mathcal{A}_i$
17: **end if**
18: **end for**
19: $g \leftarrow g + 1$
20: **end while**
21: Rank the bats and find current best $\mathbf{x}^*$
22: **return** $\mathbf{x}^*$

Algorithm 1: Bat algorithm pseudocode

where ϵ is a random number with uniform distribution on the interval $[-1, 1]$ and $\mathcal{A}^g =< \mathcal{A}_i^g >$, is the average loudness of all the bats at generation g.

If the new solution achieved is better than the previous best one, it is probabilistically accepted depending on the value of the loudness. In that case, the algorithm increases the pulse rate and decreases the loudness (lines 14-17). This process is repeated for the given number of generations. In general, the loudness decreases once a bat finds its prey (in our analogy, once a new best solution is found), while the rate of pulse emission decreases. For simplicity, the following values are commonly used: $\mathcal{A}_0 = 1$ and $\mathcal{A}_{min} = 0$, assuming that this latter value means that a bat has found the prey

and temporarily stop emitting any sound. The evolution rules for loudness and pulse rate are as follows:

$$\mathcal{A}_i^{g+1} = \alpha \mathcal{A}_i^g \quad (5)$$

$$r_i^{g+1} = r_i^0[1 - exp(-\gamma g)] \quad (6)$$

where α and γ are constants. Note that for any $0 < \alpha < 1$ an any $\gamma > 0$ we have:

$$\mathcal{A}_i^g \to 0, \quad r_i^g \to r_i^0, \quad \text{as } g \to \infty \quad (7)$$

In general, each bat should have different values for loudness and pulse emission rate, which can be computationally achieved by randomization. To this aim, we can take an initial loudness $\mathcal{A}_i^0 \in (0, 2)$ while the initial emission rate r_i^0 can be any value in the interval $[0, 1]$. Loudness and emission rates will be updated only if the new solutions are improved, an indication that the bats are moving towards the optimal solution. As a result, the bat algorithm applies a parameter tuning technique to control the dynamic behavior of a swarm of bats. Similarly, the balance between exploration and exploitation can be controlled by tuning algorithm-dependent parameters.

4. Bat Algorithm-Based Implementation for Swarm Robotics

In this section we report our implementation of the bat algorithm described above for swarm robotics by using a swarm of simple and identical robotic units. This implementation is performed at two levels: the robotic platform (physical level) and the robotic simulation framework (computational level). They are described in Sections 4.1 and 4.2, respectively.

4.1. Robotic Platform for Bat Algorithm

Our robotic swarm for bat algorithm consists of a set of identical wheeled robots. Figure 1 shows two robots of the swarm along with some of their most distinctive components. Each robot is equipped with a kit of simple yet powerful hardware components that replicate the most relevant features of the bat algorithm (the others being replicated by software). The hardware and software implementation of the robotic platform are described in Sections 4.1.1 and 4.1.2, respectively.

4.1.1. Hardware implementation

The robots were assembled by the authors on a rigid chassis, shown in green in Figure 1. The chassis and other mechanical parts (such as the wheels and holders) have been generated by 3D printing from a 1.75 mm PLA (polylactic acid) filament by using a fully enclosed domestic desktop FFF Witbox 3D printer by Spanish manufacturer QB. The robot moves through two side wheels driven by two continuous rotation servo motors. The chassis also hosts the battery holders with their board connectors (cables terminated with a standard 5.5×2.1mm, center positive barrel jack connector) and the electronics of our robotic unit. The main electronic components are:

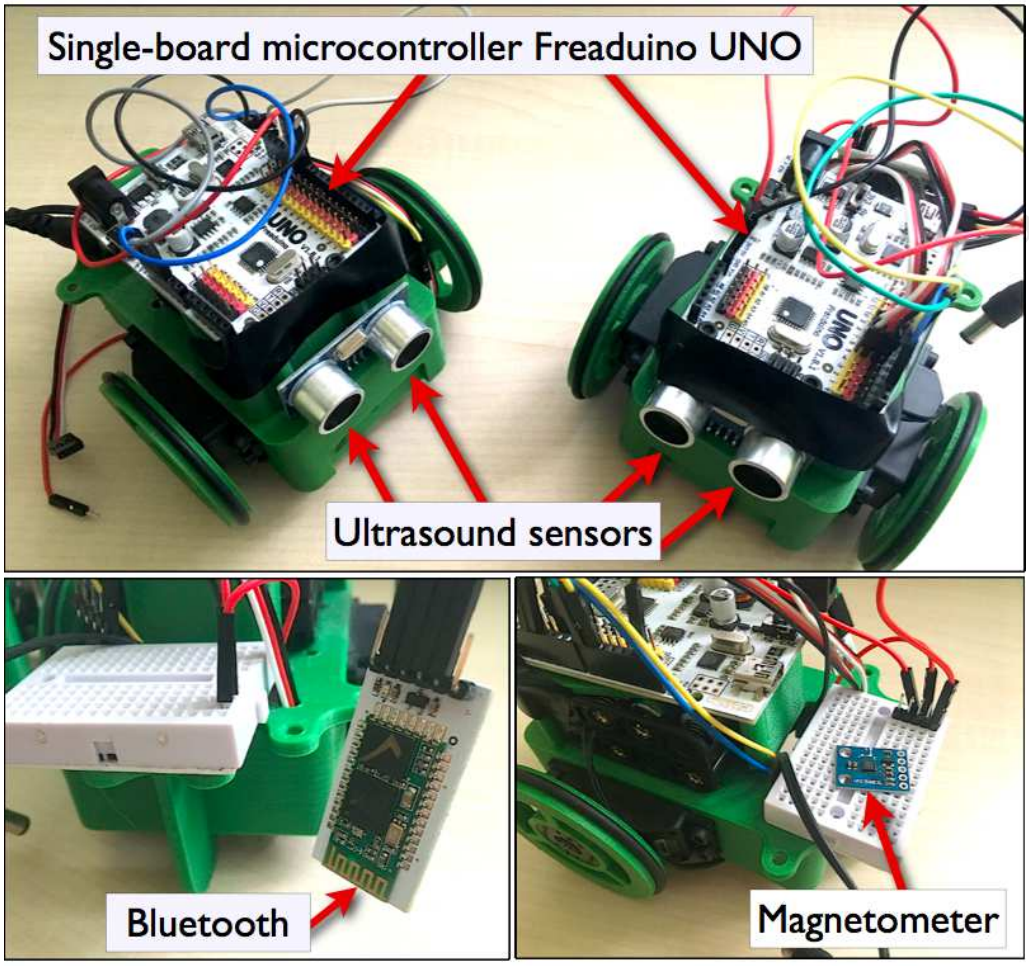


Figure 1: Identical wheeled robots used for our implementation of the bat algorithm and main components: a single-board microcontroller *Freaduino UNO* (for programming and connectivity), ultrasound sensors (for collision avoidance), magnetometer (for spatial orientation), and bluetooth card (for wireless data exchange and communication).

1. a *single-board microcontroller*: in our implementation we use the popular microcontroller *Freaduino UNO*, an *Arduino*-compatible board based on *Arduino UNO Rev3* design and manufactured by *ElecFreaks*. We choose *Freaduino UNO* because it provides a very good combination of small size, affordable price, and reasonable computing power. It is not only fully compatible with *Arduino UNO Rev3* (including all shields, programs and IDE), but also more powerful and flexible, cheaper, and easier to use. For instance, it operates at both 3.3V and 5V, while *Arduino UNO* operates at 5V exclusively. This feature allows us to connect several 3.3V modules (such as Xbee, Bluetooth transmitters, LCD screens, accelerometers, gyroscopes, and many others), as we actually do in this paper. Its range for external input voltage is wider as well: 7V to 23V for *Freaduino UNO* compared to 7V to 12V for *Arduino UNO*. It also provides a much easier

access to some valuable components: for instance, the reset button and LED are easy to press and always visible respectively, while when plug in shield in *Arduino UNO*, they turn hard to press and invisible, respectively. Furthermore, *Freaduino UNO* includes some additional interesting features with respect to *Arduino UNO*, such as the UART/IIC and SPI interface breakout and a standard mini USB connector (a type B female USB connector is used in *Arduino UNO* instead). Finally, although both boards use a single-chip microcontroller *ATmega 328* by *Atmel* (a 8-bit RISC-based microcontroller at a clock speed of 16MHz), the version in *Freaduino UNO* (*SMD ATmega 328P*) is more reliable than the *ATMega 328 (DIP)* from *Arduino UNO* board.

We found our choice to be fully suitable to meet our needs. We remark however that other microcontrollers in the market can also be used for this work (including the original *Arduino UNO* and others from the Arduino family), although additional components might be required in some cases for extra connectivity and other tasks.

The single-board micro-controller is responsible for all programming tasks and the connectivity among the different components of the robot. We also added a protoshield and a mini breadboard for further connectivity of electronic components. The protoshield is plugged on top of the board microcontroller to gain access to all microcontroller pins. The mini solderless breadboard, shown in white in Figure 1(bottom, left and right), can be used for tasks such as temporary circuits (e.g., removable sensors) and fast prototyping.

2. an *ultrasound sensor*: in this work, we use the ultrasound sensor *HC-SR04*, also manufactured by *ElecFreaks*. It is an ultrasonic sensor operating at 5V DC that uses sonar to compute the distance to an object, much like bats or dolphins actually do. Each *HC-SR04* module includes an ultrasonic transmitter, a receiver and a control circuit, with 4 pins for power, trigger (transmitter), echo (receiver), and ground. Some other technical features will be discussed in detail in Section 4.3. The ultrasound sensor is used for collision avoidance with static and dynamic objects (including other robots in the swarm) as well as with the boundaries of the physical 3D environment.
3. a *magnetometer*: in this work, we use the triple-axis magnetometer board *HMC-5883L*. This user-friendly compass is a 3.3V max chip with added circuitry to make it 5V-safe logic and power, so that it can be connected to either 3 or 5V microcontrollers. It uses I2C serial bus for easy interface to communicate. Its internal functioning is based on the anisotropic magnetoresistive (AMR) technology by *Honeywell*, with AMR directional sensors having a full range of ± 8 gauss an a resolution of up to 2 milligauss. The magnetometer is used in this work for global spatial orientation of the robotic units of the swarm regardless their physical environment.
4. a *Bluetooth card*: we use the *HC-05* Bluetooth sub-module, which uses short-wavelength UHF radio waves for wireless communication. This card is used for wireless communication and data exchange over short distances (about a range of 10 meters) among the robots of the swarm and with a central server for tracking

purposes.

All these components are connected to different I/O pins in a rather standard way. The detailed description of these connections is out of the scope of this paper and will be omitted here to keep the paper to a manageable size.

4.1.2. Sofware implementation

The robotic platform described in previous section was programmed by using several software libraries and tools. All our source code was implemented on a personal computer through the open-source *Arduino IDE*. Then, it was compiled with *AVR-GCC*, a compiler for *Atmel* AVR microcontrollers. The software library *AVR Libc* was also used for compilation. Other external libraries and files have been used for controlling the servomotors, communication via Bluetooth and with I2C devices, and other tasks. The resulting binary code (in HEX format) from compilation is transferred to the microcontroller through a RS232 serial port to TTL converter for final loading and execution.

4.2. Robotic Simulation Framework for Bat Algorithm

The bat algorithm-based robotic swarm has also been simulated graphically by using a framework specifically developed by the authors for this work. The framework has been created in *Unity 5*, a popular multiplatform game engine especially praised by its portability, supporting all personal computer operating systems and video game consoles as well as most of mobile devices and websites. This game engine comes with many different features and tools for the accurate simulation of 3D scenes, including high-quality graphics (e.g., lighting, shadows, textures, and materials) for advanced photo-realistic rendering, a powerful physics engine with realistic rigid/soft body simulations and forward/inverse kinematics, powerful navigation tools, advanced algorithms for collision detection, and so on. It also provides a nice graphical editor with different windows and workspaces for the project, scene view, game view, hierarchy, toolbars, inspector, and many other features.

Unity 5 supports the programming languages *C#* and *JavaScript*. All programming code for our robotic simulation framework has been created in *JavaScript* using the *Visual Studio* integrated programming environment (IDE). This source code can be further compiled to generate standalone applications that can be executed autonomously under different hardware configurations and operating systems. For this work, we have generated versions for Mac OS X and Windows 10, both sharing similar visual appearance and functionalities[7]. They contain three buttons to access three different pre-computed 3D scenes and a fourth button to exit the application. The 3D scenes, labelled as Level 1 to Level 3 in increasing order of geometric complexity, are displayed

[7]A preliminary version of this application was on show last February during the ICHSA 2017 conference. It was installed in a laptop publicly available to conference attendees for live execution. At that time, we received very positive comments and valuable feedback from several conference delegates. The new version keeps the same computing kernel; the modifications concern mostly to the enhanced graphical output.

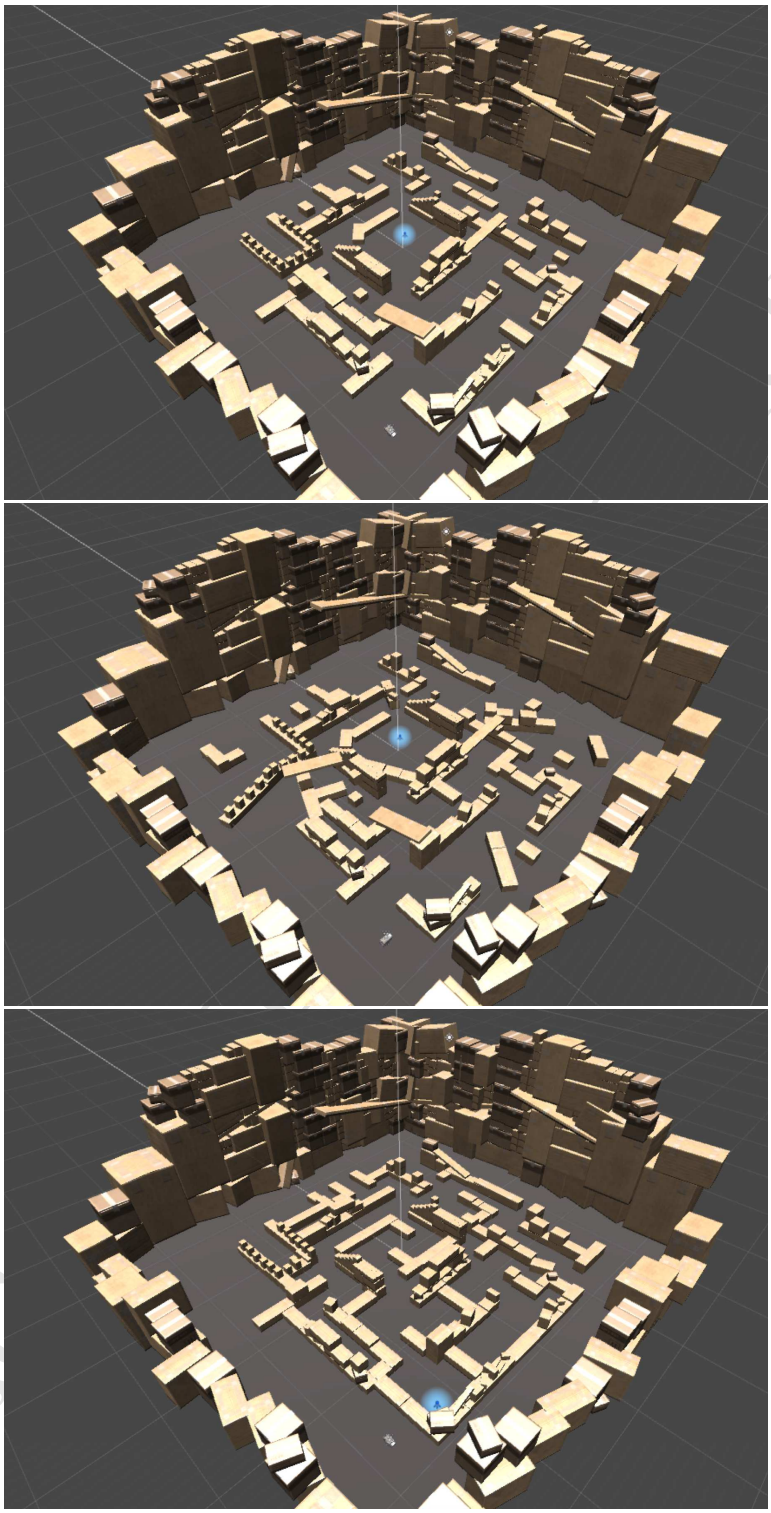


Figure 2: Graphical representation of the three 3D scenes (labelled as Level 1 to Level 3 from top to bottom, respectively).

	Real microbats	*Bat algorithm*	*Our robotic swarm approach*
No centralized behavior:	✓	✓	✓
No vision abilities:	✓	N.A.	✓
Ultrasound based:	✓	✓	✓
Operating frequency:	25–150 kHz	✓	40 kHz
# cycles burst:	10–20	N.A.	8
Operating time:	in milliseconds	N.A.	∼60 milliseconds
Accuracy range:	mm∼cm	N.A.	∼3 mm
Traveling range of pulses:	meters	N.A.	0.02∼5 meters
Loudness tuning:	✓	✓	✓
Pulse rate tuning:	✓	✓	✓
Flying abilities:	✓	N.A.	×

Table 1: Analogies and differences among the real microbats, the bat algorithm and our implementation for several features, arranged in rows (see the main text for full interpretation of the table).

in Figure 2 (top to bottom). They correspond to different spatial configurations of a closed storage room with several cardboard boxes stacked in a very messy way (even occasionally forming structures such as tunnels and passageways the robots can take advantage of). In all scenes, a blue spherical-shaped point of light is used to represent the target point, subsequently used in our experiments in Section 5. Further details about these scenes and their use for finding a target location can be found in Section 5.

4.3. Analogies of Our Implementation with the Real Bats and the Bat Algorithm

As mentioned above, all physical components of our implementation of a robotic swarm have been carefully chosen to replicate the most relevant features of the real bats and the bat algorithm by either hardware or software. This section discusses the analogies and differences found in the process. Our discussion is briefly summarized in Table 1, where we compare the real microbats, the bat algorithm, and our implementation (arranged in columns) for different features (arranged in rows). For each feature, the symbol ✓ indicates that it is supported; otherwise, the symbol × is used. N.A. stands for *Not Applicable*, meaning that this particular feature does not apply for the analyzed approach.

The most important features replicated by hardware are:

1. *No centralized behavior:* Microbats in nature neither have a leader of the group nor take centralized decisions. Instead, they synchronize their individual decisions for many important issues such as switching communal roosts, communal nursing, communal breeding, social thermoregulation (mutual warming), and so on [28]. In fact, microbats are excellent examples of non-centralized, communal decision-making. This is also a fundamental ingredient of the bat algorithm. As any

other swarm intelligence approach, there is not a centralized system controlling the swarm behavior. The same happens in our implementation. There is a central server, but it does not control the system at all; it is used for tracking and monitoring purposes only, but it does not take any decision about how to move or whatever. All robotic units move and behave on an individual (yet coordinated) basis. In our case, the bat algorithm is encoded in the microcontroller board and executed locally. The robots communicate relevant information (such as location, velocity, and frequency) to other swarm members but the dynamics of any robot is computed locally.

2. *No vision abilities:* Similar to some microbats, our robots are not provided with vision abilities at all; to this purpose, we refrain from including a camera or any other device for vision (including infrared sensors, thermal cameras, and the like). This is opposed to many usual approaches in swarm robotics, which tend to support as many sensors as possible to make more powerful robots, able to simulate a wide range of algorithms. In our philosophy, we should adhere exclusively to the principles of the bat algorithm, so this option is both unintended and unacceptable.
3. *Ultrasound based:* To faithfully replicate real bats and the bat algorithm, our robots must rely on echolocation. This is what we do in our implementation. As described in Section 4.1.1, our ultrasound sensors are based on sonar technology for collision avoidance and interaction with other robots and the environment, much like the real bats do.
4. *Operating frequency:* Although the actual range can vary depending on the microbat specie (and there are about 930 microbat species recognized so far), their typical frequencies oscillate in the range 25-150 kHz. Each ultrasound pulse of our robots operates at a constant frequency of 40 kHz, well within the range or real bats. Furthermore, it is well-known that many microbats use constant-frequency signals for echolocation, which is similar to what our robots actually do. Other microbats can use short, frequency-modulated signals to sweep through about an octave. We are currently considering options to modify the ultrasound sensor by hardware so that it can behave approximately that way as well.
5. *# of cycles of burst:* Again, this value depends on the particular microbat specie (and also on environmental conditions), with the typical range about 10-20 cycles of burst. Our robots have small ultrasound sensors sending an 8 cycle burst of ultrasound pulses, very similar to the typical value of real bats.
6. *Operating time:* Our robots capture its echo with signals lasting in the order of milliseconds, exactly as the real bats actually do. For our sensor, the manufacturer advises to consider over 60 ms measurement cycle, in order to prevent trigger signal to the echo signal.
7. *Accuracy range:* The accuracy range of the sensor is about 3 millimeters, very similar to that of some real bats.
8. *Traveling range of pulses:* in our robots this value is between 2–500 centimeters, similar to the range of a few meters of several bats.

Other interesting bat algorithm features are difficult to replicate by hardware, so they have been replicated by software to ensure the highest fidelity of our robotic swarm to the bats and the original bat algorithm. These features include:

1. *Loudness tuning:* The loudness can readily be modulated in the source programming code (as expected, since our source code is actually an implementation of the bat algorithm).
2. *Pulse rate tuning:* The pulse rate of our ultrasound signals can be modified by software to adapt to different conditions, as it typically happens with bats in the real world and in good agreement with the bat algorithm.

Finally, there an important feature of the real bats that we do not emulate in our approach: the ability to fly. While it is not really required (from a mathematical point of view) for the bat algorithm (although somehow assumed in the description of the method), this is a serious drawback towards the realism of our robotic units, especially regarding 3D environments, as they are the natural niche of microbats in real life. However, research in this area is still in its early infancy and this problem is clearly beyond the scope of this paper; more details on this issue will be discussed in Section 6.

5. Computational and Real-World Experiments

As described above, the major goal of this paper is to design and build a robotic swarm replicating the real bats and the bat algorithm as faithfully as possible. Once built, our robotic swarm is applied to some experiments about the problem of finding a target location, where the objective is to reach an unknown target location within an unknown static 3D indoor scene. This section is devoted to describe these computational and real-world experiments. The discussion starts with the description of the problem to be solved and the implementation issues in Sections 5.1 and 5.2, respectively. Then, the obtained results are discussed in detail in Section 5.3. Finally, the important issues of the parameter tuning, convergence of the method, and computational times are discussed in Sections 5.4, 5.5, and 5.6, respectively.

5.1. Problem to be Solved

In this work we consider the problem of finding an unknown target location, a problem where a swarm of n robots, $\{\mathcal{B}_i\}_{i=1,\ldots,n}$, is deployed within an unknown static 3D indoor environment, $\mathcal{S}$, at initial random locations $\{\mathbf{x}_i(0)\}_{i=1,\ldots,n}$. The goal of the swarm is to reach an unknown target location Δ within $\mathcal{S}$. We assume that the robots can freely move on a region $\Omega \subset \mathbb{R}^2$ defined as the connected set of points of the ground that are fully walkable for the robots (i.e., excluding obstacles and the boundaries of the indoor environment). We also assume that the trajectory of each robot $\mathcal{B}_i$ can be defined mathematically as a parametric curve $\mathbf{x}_i(s) \subset \Omega$, where s is the parametric variable. Therefore, if $\Delta \in \Omega$ there is always a continuous path connecting $\mathbf{x}_i(0)$ and Δ in finite time T_i, so that $\mathbf{x}_i(T_i) = \Delta$. In that case, the Euclidean distance between

these two points *on* Ω, represented onwards as d_Ω, can be computed as:

$$d_\Omega(\mathbf{x}_i(0), \Delta) = \int_0^{T_i} \sqrt{||\mathbf{x}_i'(s)||^2}\, ds \tag{8}$$

Note that the distance can only be computed on the regions where the robots actually move, discarding static and/or dynamic objects (e.g., other robots) on the way.

Similarly, at each time instance t, the distance between $\mathbf{x}_i(t)$ and Δ on Ω is given by:

$$d_\Omega(\mathbf{x}_i(t), \Delta) = \int_t^{T_i} \sqrt{||\mathbf{x}_i'(s)||^2} ds \tag{9}$$

Consider now the vector of distances $\Phi(t) = (\phi_1(t), \phi_2(t), \ldots, \phi_n(t))$, where $\phi_i(t) = d_\Omega(\mathbf{x}_i(t), \Delta)$, defined as in Eq. (9). Then, the goal is to minimize the total distance $\Xi_\Delta(t)$ of the swarm to the target Δ, that is:

$$minimize\,[\Xi_\Delta(t)] = minimize\left[\sum_{i=1}^{n}\left(\int_t^{T_i} \sqrt{||\mathbf{x}_i'(s)||^2} ds\right)\right] \tag{10}$$

In this problem, it is assumed that the robots have neither knowledge about the location of Δ nor about the possible paths leading to it. However, in order to define a proper fitness function for the experiments, each robot $\mathcal{B}_i$ is provided with the numerical value of its distance to the target point, $\phi_i(t)$.

Solving the continuous optimization problem in Eq. (11) is a difficult task for the robots, because they have no clue about the physical area around them or where to move (being thus impelled to explore the environment). For instance, it might happen that the path in front of a moving robot is blocked by an object in the environment (static obstacle), or by another moving robot of the swarm (dynamic obstacle). Since the robot cannot see the obstacle, it keeps moving forward until the distance is small enough to be detected by the short-range ultrasound sensor, at which point, different behavioral patterns can emerge (see Section 5.2). For instance, it is not always clear for the robot how to avoid the obstacle: move to the left, to the right, or back. In fact, sometimes the robot changes the plan several times in a very short time being unable to find the optimal decision (see, for instance, the robot in pink for seconds 26–30 of *Video 1*). In this situation, the use of a robotic swarm becomes very advantageous, due to its wider exploratory capacity with respect to a single robot. Ideally, each robot of the swarm explores its surrounding area and communicates its position, velocity, and frequency to the other members of the swarm, improving the global efficiency of the whole swarm to find the target.

5.2. Implementation Issues

In this paper, the experiments are performed at computational and real-world levels. Next paragraphs discuss our implementation process for both levels in detail. We

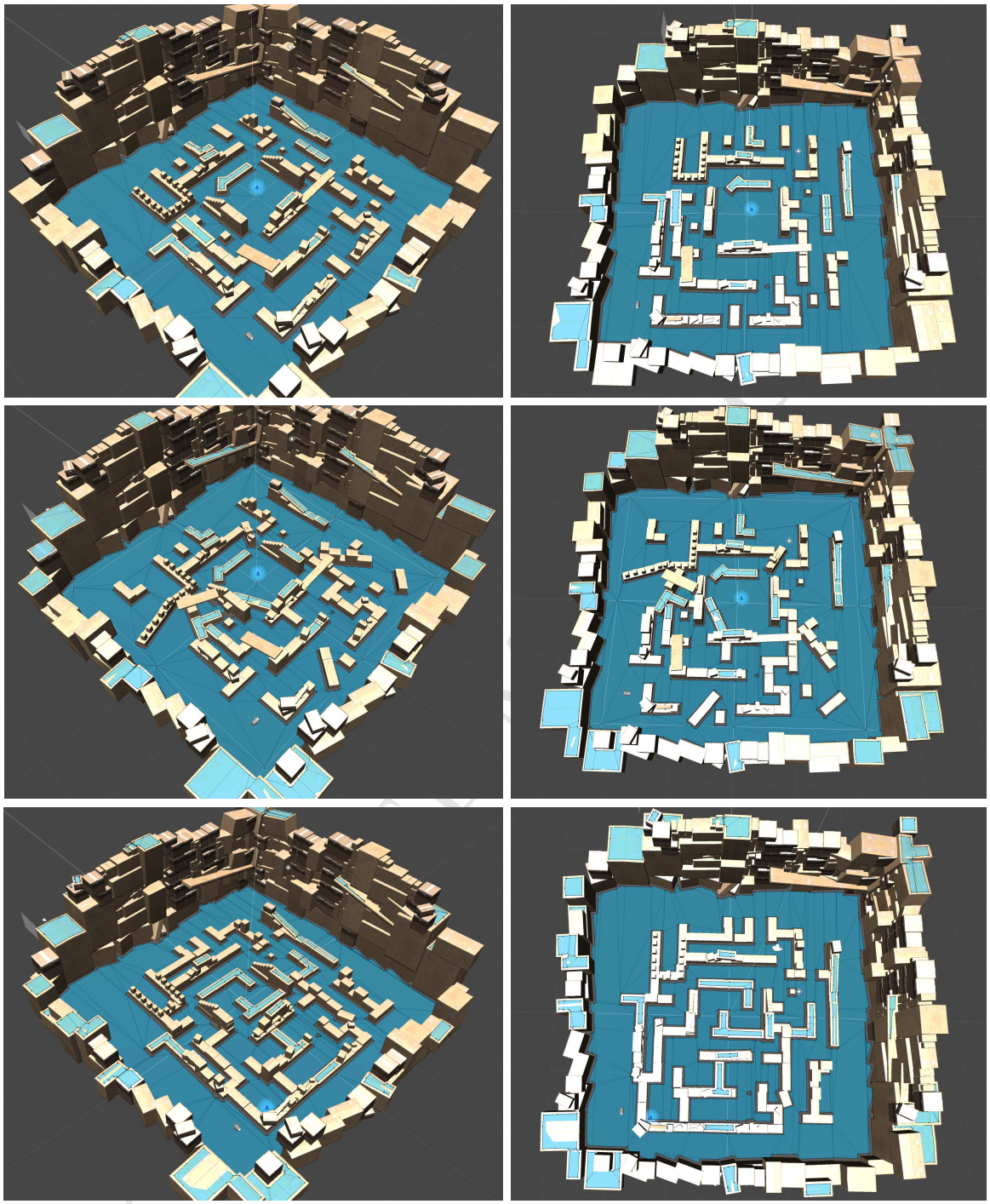


Figure 3: Navigation mesh (in green) representing the walkable areas for the three scenes (from top to bottom) in Figure 2: (left) isometric view; (right) top view.

start with the computational implementation in Section 5.2.1; then, the real-world implementation is discussed in Section 5.2.2.

5.2.1. Computational implementation

In order to apply the bat algorithm to the continuous optimization problem in Eq. (11), we firstly need to determine some definitions in Section 5.1 in our implementation. For instance, it is important to determine the walkable areas of the scenes for the computational experiments. Our game engine is particularly useful in this regard: *Unity 5* provides powerful options to compute and handle such areas through a tool called *navigation mesh* (or *navmesh*), a collection of non-overlapping two-dimensional convex polygons defining areas of the map fully traversable by the robots. Figure 3 shows the navigation meshes (in green) associated with the three scenes (from top to bottom) in this paper. We show both the isometric view (left) and the top view (right) for full visualization of all scene features. Note that the different objects in the scene (mostly cardboard boxes piled up in a messy way) are surrounded by a security area (in vanilla color) defined by a threshold value of the normal distance from the object. This is used to prevent collisions between the objects and the moving robots. It corresponds to a threshold value for the ultrasound sensor in real-life experiments, which can be set as low as 2cm and as high as 400cm.

Once the navigation mesh of the scene is obtained, it is stored in memory to be subsequently used for the objective function. In our computational experiments, we compute the distance between the current position of each robot at iteration step j and the target location as: $\phi_i^j = d_\Omega(\mathbf{x}_i^j, \Delta)$, where the superscript j is used to indicate the iteration. Then, we consider the vector of discrete candidate solutions $\Phi^j = (\phi_1^j, \phi_2^j, \ldots, \phi_n^j)$. From it, the fitness function to be minimized is given by:

$$\Xi_\Delta^j = \sum_{i=1}^{n} \phi_i^j = \sum_{i=1}^{n} d_\Omega(\mathbf{x}_i^j, \Delta) \tag{11}$$

where ϕ_i^j is calculated as the total length of a polyline connecting $\mathbf{x}_i^j$ and Δ.

All bats are initialized at random positions on the domain Ω. However, we found that even a small population of about 10 bats might be excessive for a scene with a relatively small area, where the initial locations for some robots might be very close to the target point. Therefore, to make the problem more challenging, we initialize the robots in the areas farthest from the target. Such areas are then subdivided in smaller non-overlapping rectangular regions of size $[u_a, u_b] \times [v_c, v_d]$ on the parametric domain Ω, and the robots are randomly initialized with a uniform distribution on such regions. Of course, the number of robots and the number of regions are not equal in general; as a result, depending on such numbers, some regions might be empty while others can host more than robot at once. This initialization strategy ensures that all search space can be visited while preventing solving the problem too quickly for small area (or highly populated) scenes.

For our computational experiments, we generated two similar standalone applications for both Mac OS X and Windows 10, each running on a different computer. In the first case, we use a Apple iMac Pro, with a 4.2 GHz quad-core Intel Core i5, with 8GB of 2400MHz DDR4 memory, 2TB Fusion Drive hard disk and a graphical card Radeon Pro 580 with 8GB of VRAM. For the Windows version, we use a 3.8 GHz quad-core

Intel Core i5, with 16 GB of DDR3 memory, and a graphical card AMD RX580 with 8GB of VRAM.

5.2.2. Real-world implementation

Our real-world experiments have been carried out by using a swarm of 10 real robots distributed on a real-world indoor environment. The environment was a closed storage room in our department used to store many cardboard boxes of our devices (computers, printers, photocopying machines, and so on) and other stuff (paper, toner, etc). The storage room is no longer in use, and the boxes are stacked in a very messy way. These features make it a challenging yet controlled environment to carry out our real-world simulations. However, some previous work was needed to make the environment fully usable. After cleaning the room, we modified the spatial configuration of the boxes to create several complex and irregular paths where the robots can really move. We then created a map of the room for this new spatial configuration. This map has been generated in our computer simulation framework by using handheld electronic distance measurement (EDM) devices from the surveying department (similar to ours, located in the Civil Engineering School of our university). In this way, we can register all corners in the scene to define the navigation mesh for the robots with high accuracy. As a result, three different configurations have been generated. They correspond to the three maps displayed in Figure 2.

The next step concerns the deployment of the robots in this environment. Firstly, we place the robots in the scene at will and record their actual position through triangulation (by using the global positions of the surrounding corners, already registered in our computer model of the scene). Then, to make sure that the robots are initialized at random locations, we generate a random distribution by computer in the simulated map of the scene, a very accurate replica of the real room. These initial random locations are transmitted to the robots via wireless communication through the Bluetooth card. Then, the robots move automatically to their initial locations using the magnetometer for spatial orientation when following a precomputed route from the initial point where we placed them. At the end of the process, all robots are located at the generated random locations. The magnetometer also allows us to define the orientation of the robot, determined by the angle with respect to the North magnetic pole of the Earth. This value is also randomly initialized. All these random values are also saved in a file to be subsequently used in the computer simulations, thus ensuring that both the computational and the real-world experiments are carried out for the same initial conditions.

5.3. Our Results

A series of experiments have been conducted at both computational and real-world levels. For the former, we carried out series of 200 computer executions for each of the three different scenes displayed in Figure 2, where the initial locations of the robots are set randomly for each individual execution. Regarding the real-world level, we carried out just 20 simulations instead of 200 because of battery constraints, using the three scenes. Unfortunately, the large size (with respect to the robot size) of the real room

Figure 4: Computational and real-world experiment about the problem of finding a target location for swarm robotics: general view (top-left), top view (top-right) and selected regions of interest (bottom-middle & right).

and the boxes within do not allow us to take wide range pictures, only close-up (mostly uninformative) photos. Due to this reason, we rely on computer simulations to show our results.

First and most important observation of our experiments is that *the behavior of the robotic swarm is amazingly similar for both the computer and real-world experiments*, provided that we use the same scene, number of robots, and initial locations. For illustration, Figures 4 and 5 shows two screenshots of one execution of the computer experiment for the second scene along with the pictures of its real-world counterpart. We selected these particular screenshots to show some interesting features for comparison between the computer and real-world simulations. Figures 4 and 5 share a similar structure: the upper part of the figure shows the computer simulation of the full scene with isometric view, while the image on the top-right part shows the top view. Ten robots were deployed in each scene with initial random locations and they evolved according to the bat algorithm, as described above. The virtual robots were graphically generated by the authors in open-source modeler *Blender* from real pictures and then exported to *Unity* for realistic graphical representation in the computer simulations.

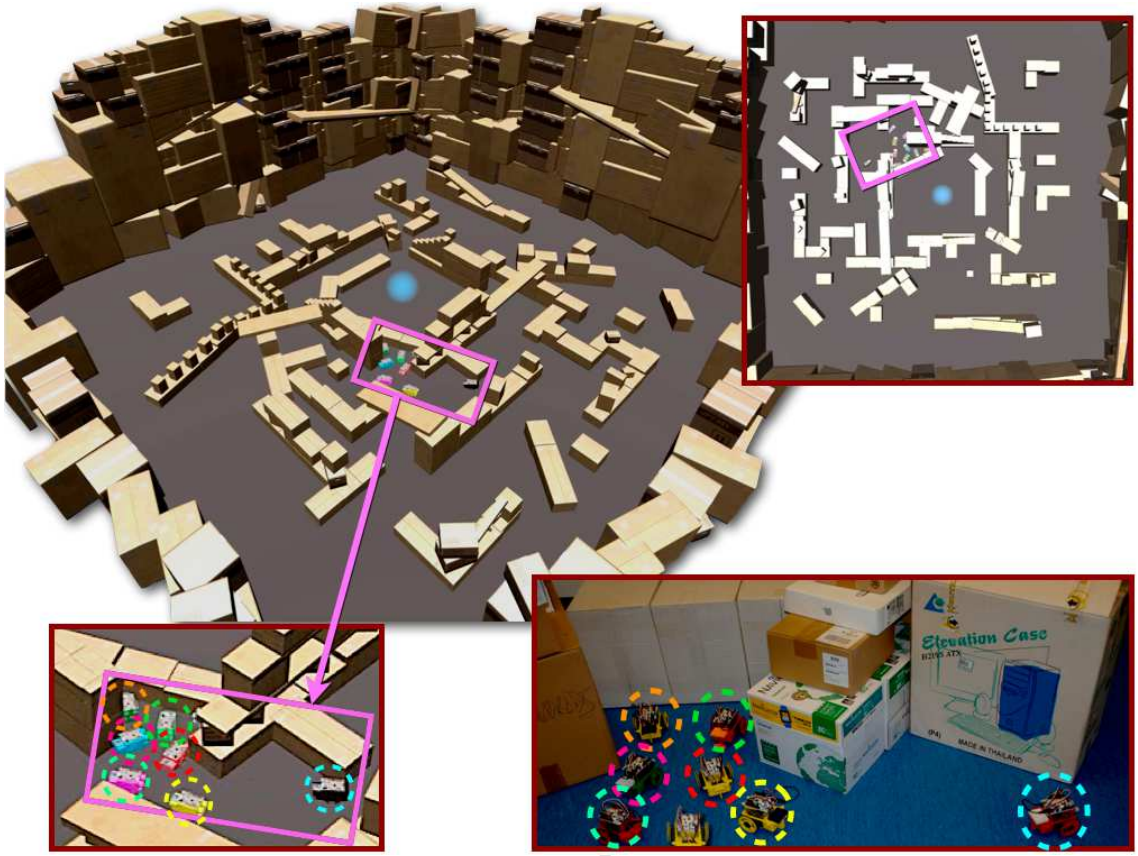


Figure 5: Computational and real-world experiment about the problem of finding a target location for swarm robotics: general view (top-left), top view (top-right) and selected regions of interest (bottom-left & right).

They are also textured and displayed in different colors for easier identification. We remark, however, that their colors do not match in general those of the real robots (unfortunately, we had only three colors available for the 3D printed robotic chassis: green, red and yellow). A rectangle (in red and pink in Figs. 4 and 5, respectively) shows a selected region of interest (ROI) in both pictures. These regions are further enlarged (as indicated by arrows of similar colors) for better visualization. The ROIs contain some robotic units highlighted with dashed circles of different colors in the images for easier identification. We also show a picture of the same real-world experiment with the same robots for better visual comparison.

We remark that the computational and the real-world simulations are performed in a completely independent way, but for the same scene and the same initial locations for the real and the simulated robots. We noticed a very similar behavior of the robotic swarm in both experiments. Their positions and motions match very well, as shown in those pictures.

Another important observation is that *the evolution of the robotic swarms in computer and real-world experiments* (albeit globally very similar) *is not really identical*.

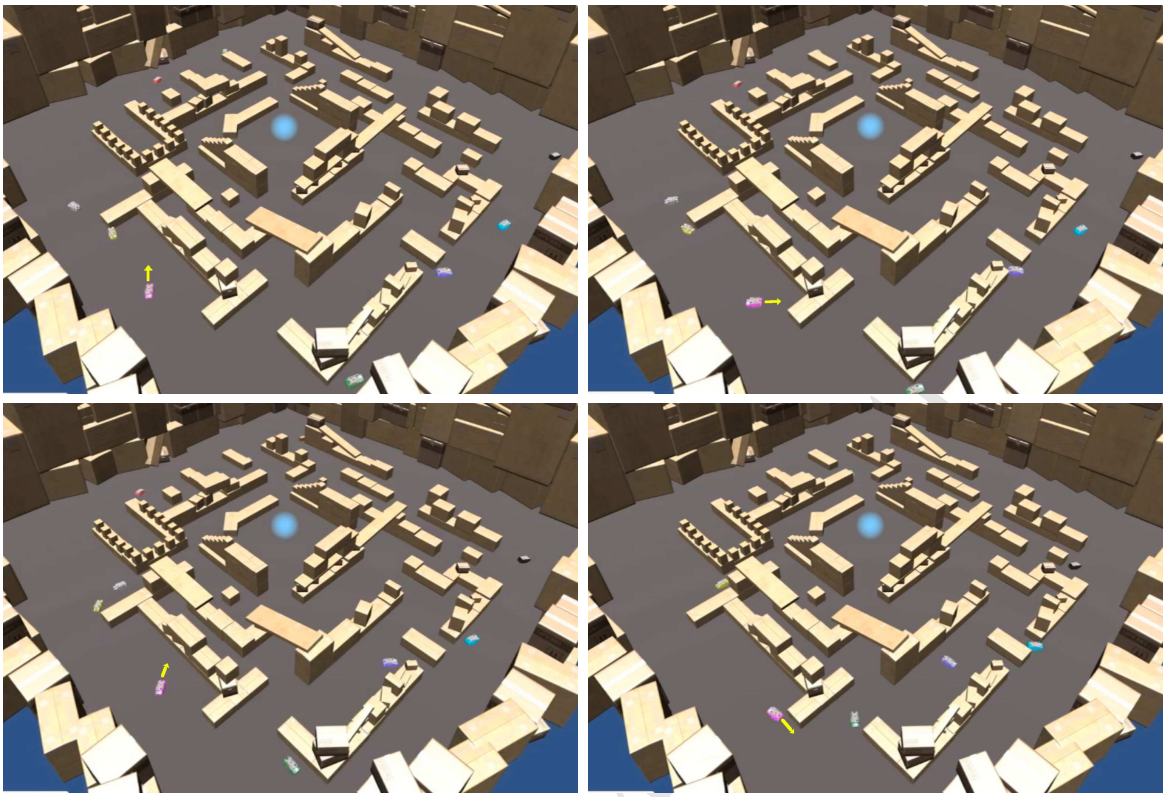


Figure 6: (top-bottom, left-right) Four screenshots of Video 1 showing some interesting behavioral patterns (see the main text for details). The pictures have been edited for arrows.

In our experiments, we noticed that the actual positions of the real and the simulated robots do not match exactly frame-by-frame during the simulation timeline, because of several factors. In some cases, the real-world robotic swarm is advanced or delayed with respect to the computer robotic swarm. In other cases, the evolution of the robots is not exactly the same, probably due to technical factors (such as the sensor sensitivity, a different position of an object in the computational and real-world scene, or others). Yet, the general dynamics of the swarm is surprisingly similar in most cases.

Our experimental results have shown that the bat algorithm is very well suited for this problem, actually much better than expected. We found a number of interesting behavioral patterns to be further explored. Some of them are reported in the accompanying material submitted with this paper. It consists of six short videos, numbered from 1 to 6, corresponding to the three scenes of the paper (videos 1 & 2 for the first scene, videos 3 to 5 for the second scene, and video 6 for the third scene). The images in the videos show the screenshots recorded from a single illustrative execution. The window is divided into three parts, with the main (isometric) view on the left, the top view (top-right) and the convergence diagram of the different robots (bottom-right).

Video 1 shows some illustrative examples about the difficulties of some robots to decide the direction to take in order to approach the target location. This is clearly shown by the robot in pink on the left at the beginning of the simulation. The robot

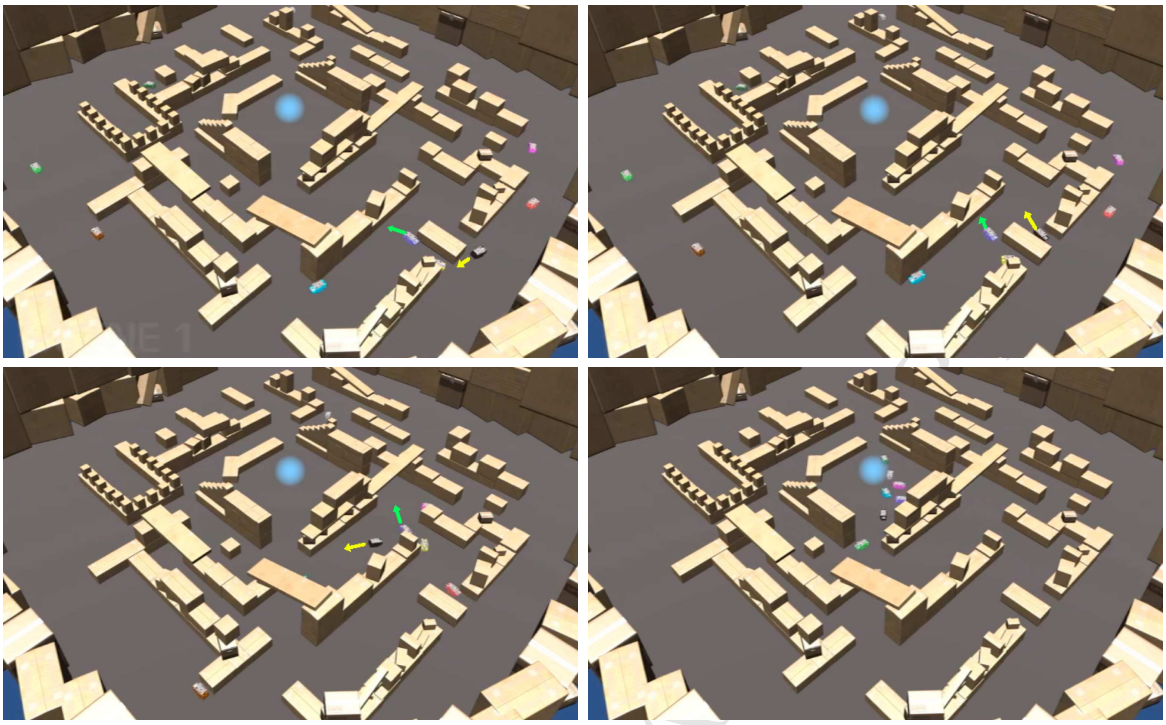


Figure 7: (top-bottom, left-right) Four screenshots of Video 2 showing some interesting behavioral patterns (see the main text for details). The pictures have been edited for arrows.

moves to the left and right several times in sequence, unable to determine the best strategy for a while. This behavior is graphically captures in four screenshots in Figure 6, corresponding to the seconds 26, 27, 28 and 30 of the animation. A similar wandering motion pattern can be found for the robot in purple in lower middle part. The video also shows the ability of the robots to find the target location through different paths. Similar behavioral patterns can be found for the second scene in *Video 3*.

Video 2 shows an interesting example of a bifurcation behavior, represented in the four screenshots of Figure 7. Initially, the robots in black and purple take two different directions, represented respectively by the yellow and green arrows (top-left), but they suddenly change their directions (top-right) while advancing towards the target. Then again, they interchange their directions (bottom-left), constantly seeking for the most promising path to the target until finally reaching their goal (bottom-right). Similar (and even more intriguing and complex) behavioral patterns can be found for the second scene in *Video 4*.

Video 5 shows several sophisticated and complex behavioral patterns. For instance, in our prior opinion, the robots should get trapped in challenging configurations such as U and V shapes (where the robotic units would be forced to come back), in cases of traffic jam owing to overcrowding, or in cases of narrow passages and corridors where only a single robot could advance at once. However, the swarm always find a way to escape from these configurations. The video shows two very illustrative situations,

which are also displayed in Figures 4 and 5. In the first case, the first robot of the formation enters into a narrow dead end, while the next robots in the formation simply follow it. At the end, they get trapped inside and, because of the collision with the boundaries of the corridor and amongst the robots of the swarm, they cannot go back anymore. However, the robots at the queue of the formation are actually outside the corridor, so they can turn around and look for an alternative path. As soon as the first moving robot approaches to the target, it becomes the new global best, thus attracting other members of the swarm to these new promising areas of the search space. In the second case, the formation gets cornered in an irregular convex U shape. The robots in the formation try to turn left and right in order to get rid of this trap, but are affected by the constraints given by the collisions with their neighbors. However, these local perturbations can eventually lead to some robots to find a escape route, a kind of local search method that prevents the robots to get trapped in these challenging shapes. A similar situation can be found for the third scene in *Video 6*. That video is also very helpful in showing that the robots do not actually know the path leading the target. Take the robot in pink for a very illustrative example: initially, the robot moves towards the swarm, which is gathering in a location far away from the target. At some point, the robot turns back and moves in the opposite direction, approaching the target until reaching a bifurcation point where the path splits. Suddenly, the robot chooses the wrong way. However, since the other members of the swarm are very far from it, this robot is still the global best and hence, the formation follows its way, even if that means to get trapped in a dead end room. After a while, some members of the swarm find a way to exit the room, and approach the target location, dragging the rest of the swarm with them until eventually reaching the target. Other interesting behaviors have also been found in other videos and executions. They are omitted here to keep the paper and accompanying material at reasonable size.

5.4. Parameter Tuning

A critical issue when working with nature-inspired metaheuristic methods is the parameter tuning. It is well-known that the performance of these methods is strongly dependent on the choice of suitable values for their parameters. Moreover, such values are problem-dependent, making it hard to determine good values in advance. Therefore, although there are some papers describing suitable values for some problems, our choice must be necessarily empirical. To this purpose, we carried out numerous computer simulations for different parameter values. The different parameters used in this paper are arranged in rows in Table 2. For each parameter, the table shows (in columns) its symbol, meaning, range of values, and the parameter value chosen in this paper.

The most important parameters in the bat algorithm are:

- *population size*: in general, increasing the number of individuals decreases the number of required iterations, but it also increases the number of function evaluations. Therefore, a trade-off between both situations must be achieved for better performance. In this work, we tested populations ranging from 10 to 20

Symbol	*Meaning*	*Range of Values*	*Selected Value*
$\mathcal{P}$	population size	$10 - 20$	10
g	maximum number of generations	$100 - 300$	200
$\mathcal{A}^0$	initial loudness	$(0, 2)$	0.5
$\mathcal{A}_{min}$	minimum loudness	$[0, 1]$	0
r^0	initial pulse rate	$[0, 1]$	0.25
f_{max}	maximum frequency	$[0, 10]$	2
α	multiplicative factor	$(0, 1)$	0.5
γ	exponential factor	$[0, 1]$	0.4

Table 2: Bat algorithm parameters and their values in this paper.

robots and found that increasing the number of individuals reduces the required time for convergence, but some of the interesting behaviors reported above were lost because of fast convergence (with 15–20 robots in a relatively small room, there is a large probability that at least one is randomly initialized very near to the target). Therefore, we finally decided to set this value to 10 robots for comparative purposes with the real-world experiments, in which we use actually 10 robots.

- *maximum number of iterations*: we tested our method for values of this parameter in the range $100 - 300$ and found that the method converged in less than 200 iterations in all our executions, so we finally set this parameter to 200 iterations. To prevent wasting computation time, we also stop the execution when the swarm achieves convergence (i.e., all units of the swarm reach the target point).

- *initial and minimum loudness and parameter* α: they are set to 0.5, 0, and 0.5, respectively. However, from our computer experiments we noticed that our results do not change significantly for values of the initial loudness in the whole range $(0, 2)$, meaning that this parameter is very robust against variations on that interval. We used $\alpha = 0.5$ as suggested in some previous papers, but did not check other values for this parameter in detail.

- *initial pulse rate and parameter* γ: of these two parameters, the initial pulse rate is the most relevant. In fact, parameter γ only affects the very early iterations. We set their values to 0.25 and 0.4, respectively.

With this choice of parameter values, we run the bat algorithm iteratively. Positions and velocities of the robots are computed and ranked according to our fitness function explained above. The iterative process stops once either all robots of the swarm have arrived at the target point successfully or the maximum number of iterations is reached, whatever comes first.

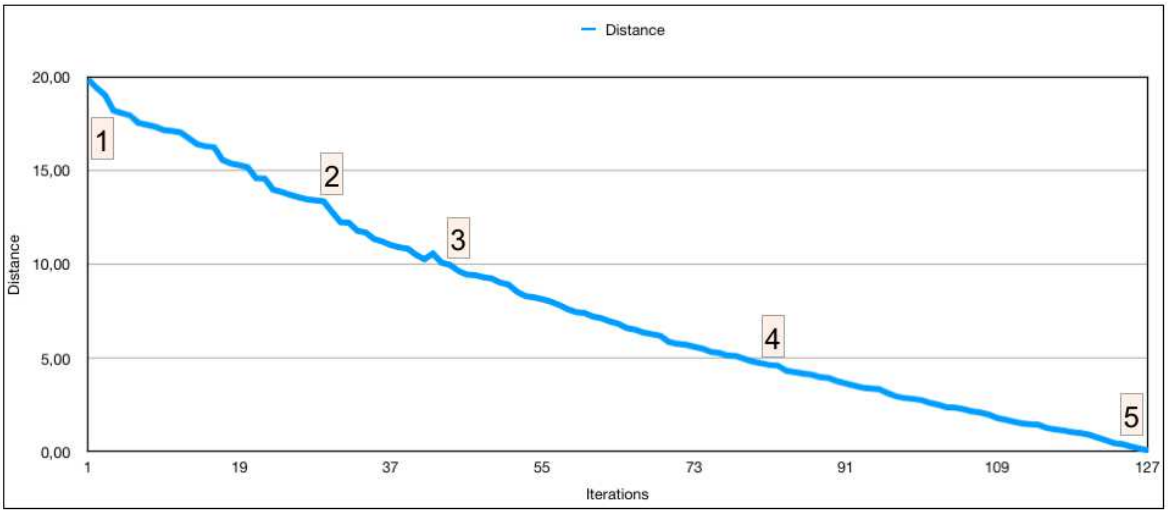


Figure 8: Convergence diagram for the execution in Video 5 and Figures 4 and 5.

5.5. Convergence Issues

An important issue of our approach is the convergence of the robotic swarm to the target point. To test this issue, we modified the spatial configuration of the three scenes in this paper in many different ways (adding new obstacles in the map, moving the location of some objects, removing objects, and so on) to get hundreds of variations of different geometric complexity.

After thousands of simulations, we did not find any configuration unsolvable for the robots so far, provided that at least one 2D solution exists. Our experiments show that the bat algorithm is very well suited for this task. It also shows that our implementation for robotic swarms performs extremely well, actually much better than we anticipated.

Of course, the convergence time of the process can vary a lot for each individual execution, depending on the geometry of the scene, initial location of the robots, population of the swarm, and other factors. Just for illustration, Figure 8 shows the convergence diagram for a single execution on the second scene displayed in Video 5 and in Figures 4 and 5. The diagram displays the numerical value of the distance to the target for the best member of the swarm (vertical axis) over the number of iterations (horizontal axis). Five tags at different times are used to indicate different stages of the process, represented graphically by screenshots in Figure 9 for better visualization: initial random distribution of the robots in the scene, high value of fitness function (stage 1); initial swarming around the current global best without a clear idea about where the target is, fitness function decreasing steadily but very slowly (stage 2); approaching to (but still far from) the target, then the formation gets temporarily trapped in a narrow dead end, so the fitness function even increases slightly (stage 3); the swarm is slowly approaching to the target, until getting trapped in a U-shape configuration for a while, then the group finds a way to escape from it and the fitness function decreases again (stage 4); the swarm advances until achieving the target point, fitness function decreases until convergence (stage 5).

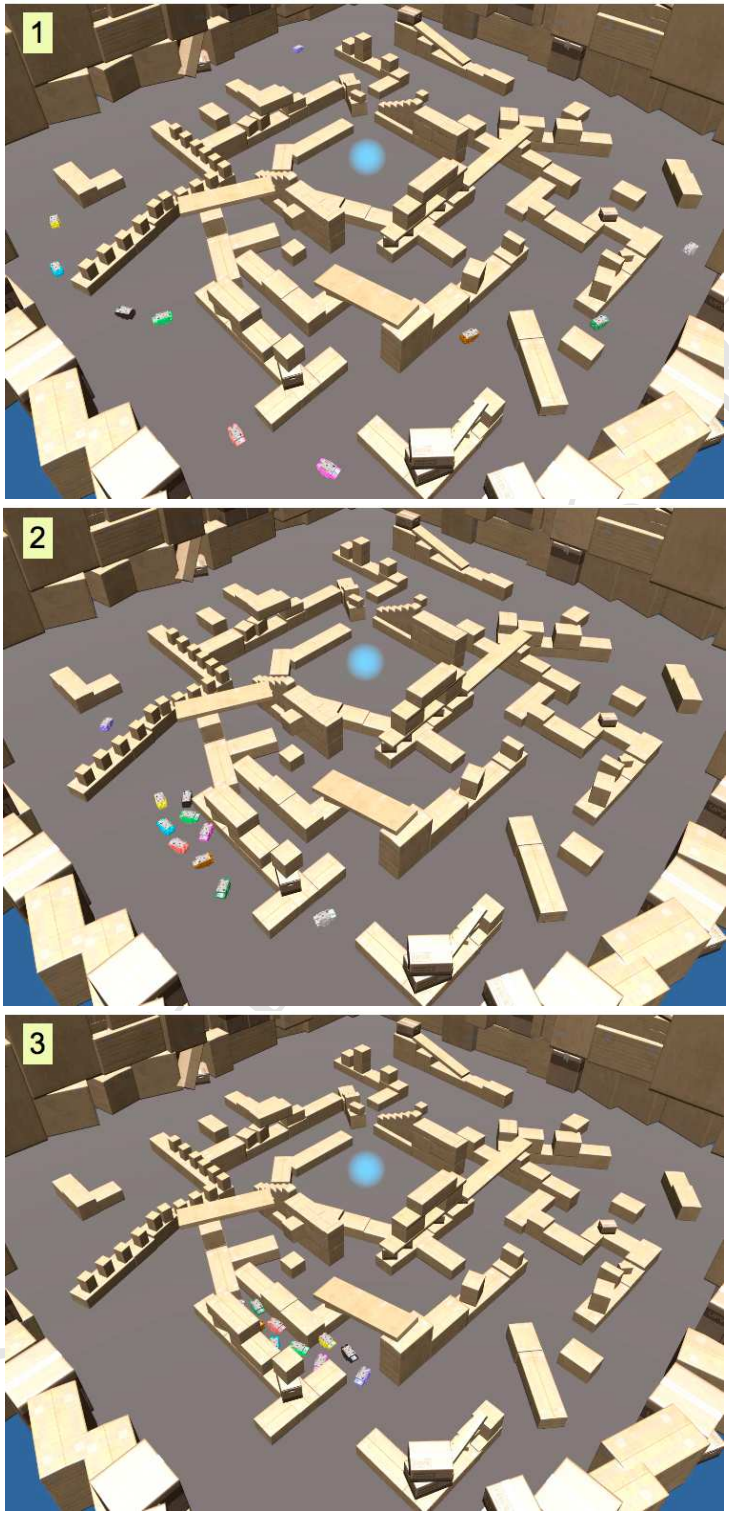


Figure 9: Individual pictures of the stages in the convergence diagram of Figure 8: (top) stage 1; (middle) stage 2; (bottom) stage 3.

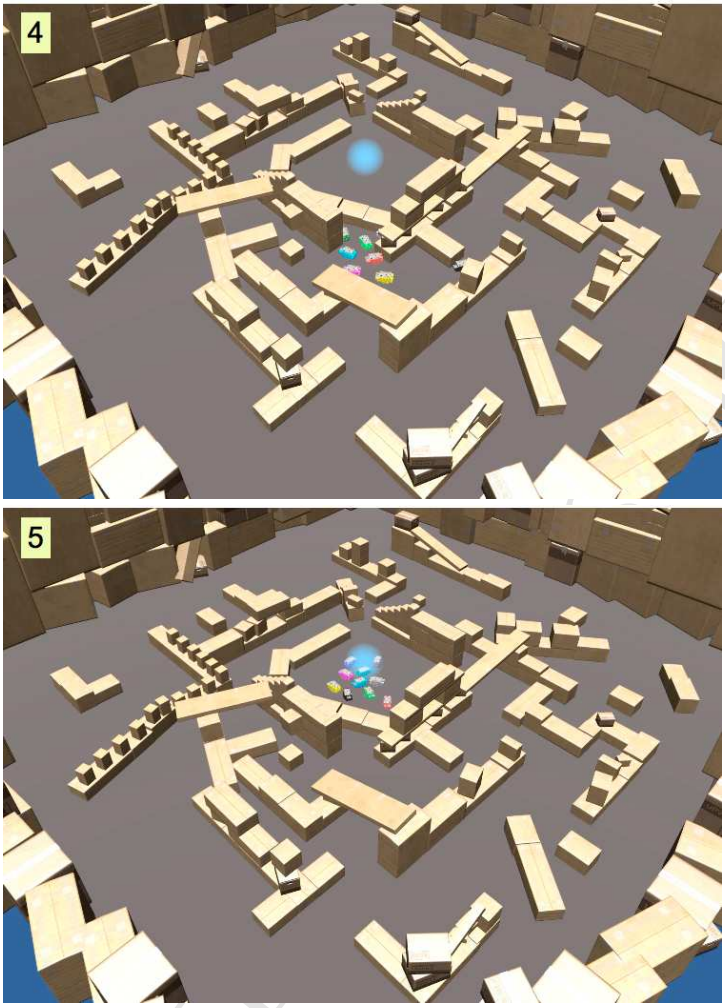


Figure 9: *(cont'd)* Individual pictures of the stages in the convergence diagram of Figure 8: (top) stage 4; (bottom) stage 5.

5.6. Computational Times

Regarding the computational times, Table 3 reports our results on a total of 200 independent executions per model. For each scene (arranged in rows), the table reports the following items (in columns): the average CPU time (measured in seconds) taken by the bat algorithm to finish the execution (excluding rendering, collision detection, and other issues); the average CPU time to finish all processes; the range of minimum and maximum values for the number of iterations along with the mean value; the percentage of success and the average CPU time for each bat algorithm iteration.

First observation is that the bat algorithm performs very well in terms of CPU time. All executions are very fast, typically taking only a few hundredths of a second. This can be explained by two factors: the small population of bats used in this work, and the low number of iterations required for convergence. Also, from the third column in the table it becomes clear that most of the CPU time in our system is used for tasks

Model	Average Time (in sec.)		# of Iterations		% of	Average Time
	Only BA	*rendering*	*Range*	*Mean*	Success	of BA Iteration
Scene 1	1.793E−2	2.107	13–82	43.8	100	4.093E−4
Scene 2	2.016E−2	2.598	21–163	47.4	100	4.253E−4
Scene 3	3.401E−2	6.872	39–187	81.7	100	4.162E−4

Table 3: Computational times for the scenes used in this paper.

other than the bat algorithm, mainly collision detection, distance determination, and rendering. Collision detection is a potential bottleneck for highly populated swarms, as the potential number of collisions scales as $O(N^2)$ in the worst case (i.e., without any heuristics to reduce this load), N being the number of robots. On the other hand, rendering can take a significant time with respect to the bat algorithm, but this is something difficult to determine as it also involves the GPU in addition to the CPU (see Section 5.2.1 for the description of the hardware used in this work). Anyway, even with all these processes, a typical run takes only a few seconds to finish. The process is so fast that it is difficult to follow at real speed. In fact, we had to pause our executions tenths of a second for each iteration during the production of our videos to allow our viewers to grasp the visual details of the animations.

6. Comparison with Other Alternative Approaches

It is a common practice in swarm robotics to carry out comparative analysis about the performance of a robotic swarm implementation for different swarm intelligence methods. But while this question makes full sense for general-purpose robotic units, it does not apply to the implementation introduced in this paper. Since our robotic units are specifically designed for the bat algorithm, we cannot expect a good performance for any other swarm intelligence method. The real comparison should be done with other robotic implementations for the bat algorithm or its variations.

As we previously mentioned in Section 1.4, to the best of our knowledge, this is the first practical implementation of the bat algorithm for swarm robotics. This fact prevents any comparative work with previous approaches in the field. It is not, however, the first implementation of robots to replicate relevant features of real bats. A very recent paper introduced *Bat Bot* [38], an autonomous flying robot aimed at replicating the flight of real bats in a physical environment[8]. Aerial robots based on insects and birds had already been described in the literature, but the bats are particularly challenging because of their complex body skeleton (compared to birds and insects) and highly irregular flying patterns. The wings of *Bat Bot* are simplified versions of the original: instead of over 40 joints of the original bats, *Bat Bot* contains only nine joints (five of them controlled by mini-motors and four that are merely passive) made of lightweight carbon fibre, combined with an ultra-thin silicone membrane that

[8]Visit also the URL: *http://resolver.caltech.edu/CaltechAUTHORS:20170201-201024724* for very interesting supplementary material, including some videos, about that work.

mimics the bat skin covering each wing. As a result, the robot weighs only 93 grams and is controlled using tiny motors in its backbone. It also has onboard sensors to measure the angle of the joints for adjustment during the flight. This design brings significant improvements over current aerial robots in terms of energy efficiency due to their articulated soft wing architecture, and the fact that wing flexibility amplifies the motion of the robot's actuators. Complex movements involving asymmetric wing folding of the main flexible wings to control the heading of the robot are also possible. Furthermore, the *Bat Bot* does not use high-speed rotors, making it less noisy and intrusive than other flying robots. Note, however, that the design is exclusively focused on replicating the flying pattern of bats. In other words, it does not address any issue about the collective behavior of bats as a swarm. Although several *Bat Bots* could fly on the same environment, they neither communicate nor cooperate together for a common task. As impressive as it is, it is not actually related to swarm robotics or the bat algorithm in any way.

7. Conclusions and Future Work

In this paper we introduced the first practical implementation of the bat algorithm for swarm robotics. Our implementation is performed at two different and complementary levels: a physical real-world level, where we design and build a real robotic prototype; and a computational level, where we develop a robotic simulation framework for the bat algorithm in swarm robotics. A very important feature of our implementation is its high specialization at both levels. Our implementation has been carefully designed to replicate the most relevant features of the real microbats and the bat algorithm as faithfully as possible. To this purpose, all our (physical and logical) components are fully optimized to follow the bat algorithm principles as much as possible. The paper describes our implementation at both levels in enough detail so that any interested reader can replicate our approach with reasonable effort. We also discuss the analogies and differences among the real microbats, the bat algorithm, and our approach.

Our implementation has been tested by its application to a problem of the problem of finding a target location within unknown static indoor 3D environments. Our experimental results show that the behavioral patterns observed in the real and the simulated robotic swarms are very similar. In most cases, the behavior and evolution of the robotic swarms at the physical and computational levels match each other fairly accurately. They are not identical, however, due to some technical and environmental issues. From our experimental results, we have been able to identify several interesting behavioral patterns. They show that the robotic swam units follow some kind of (non trivial and difficult to grasp at first sight) intelligent behavior in order to solve the intended problem. This makes our robotic swarm implementation an ideal tool to explore the potential and limitations of the bat algorithm for real-world practical applications and their computer simulations (actually one of the main purposes of this research).

This work can be extended in many different ways. On one hand, although some behavioral patterns have already been discussed here, they probably represent only a

small subset of all possible patterns to be discovered and analyzed. Several strategies for additional experiments (such as modification of the current 3D scenes, consideration of other 3D locations, along with other different problems) can arguably lead to the emergence of a bulk of new, exciting intelligent behavioral patterns for the swarm. One of the reviewers suggested us to link the observed patterns and behaviors to the reasons behind why the bat algorithm allows them happen and to identify the relevant parameters involved in such processes. We certainly agree with the reviewer that this is a very relevant question, and are very interested to analyze this issue in deep detail as part of our future work in the field.

On the other hand, we would like to improve our physical robotic units towards their miniaturization. Better physical components (sensors, actuators, etc.) would arguably improve the performance of the robotic swarm, although is it still not clear to us at what extent. A critical issue towards the realism of our robotic swarm for the bat algorithm concerns its ability to fly. This is currently a very hot field of research. The most recent advances on this subject open the door for more realistic implementations of bat-based flying robots, which might be paired with swarm intelligence for better performance. We would like to investigate further on this issue. The development and deployment of larger robotic swarms, in line with recent research work in the area of swarm robotics, is also part of our plans for future work.

Accompanying Material

In addition to this paper, the reader is kindly invited to take a look at the six videos submitted with this work. The videos, in MPEG-4 format, can be reproduced with popular media players such as *VLC*, *QuickTime*, *KMP*, and many others.

Acknowledgments

This research has been kindly supported by the Computer Science National Program of the Spanish Research Agency (Agencia Estatal de Investigación) and European Funds, Project #TIN2017-89275-R (AEI/FEDER, UE), the project EVOLFOR-MAS Ref. #JU12, jointly supported by public body SODERCAN of the Regional Government of Cantabria and the European funds FEDER, the project PDE-GIR of the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie Actions grant agreement #778035, Toho University (Funabashi, Japan), and the University of Cantabria (Santander, Spain). The authors are particularly grateful to the Department of Information Science of Toho University for all the facilities given to carry out this work. Special thanks are also due to the Editors and the three anonymous reviewers for their encouraging and constructive comments and very helpful feedback that allowed us to improve our paper significantly.

Bibliography

[1] Arkin, C.R.: *Behavior-Based Robotics*. MIT Press, Cambridge, Mass, USA (1998).

[2] Arvin, F., Samsudin, K., Ramli, A. R.: Development of a miniature robot for swarm robotic application. *International Journal of Computer and Electrical Engineering*, **1**, 436–442 (2009).

[3] F. Arvin, F., Murray, J.C., Shi, L., Zhang, C., Yue, S.: Development of an autonomous micro robot for swarm robotics. In: *Proc. IEEE International Conference on Mechatronics and Automation*, Tianjin, 635–640 (2014).

[4] F. Arvin, Yue, S., Xiong, C.: Colias-Φ: an autonomous micro robot for artificial pheromone communication. *International Journal of Mechanical Engineering and Robotics Research*, **4**(4), 349–353 (2015).

[5] Blum, C., Merkle, D. (eds.): *Swarm Intelligence: Introduction and Applications*. Natural Computing Series. Springer Berlin Heidelberg (2008).

[6] Bonabeau, E., Dorigo, M., Theraulaz. G: *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, New York (1999).

[7] Caprari, G., Balmer, P., Piguet, R., Siegwart, R.: The autonomous micro robot "Alice": a platform for scientific and commercial applications. *Proc. of Int. Symposium on Micromechatronics and Human Science, MHS '98*, 231–235 (1998).

[8] Caprari, G., Estier, T., Siegwart, R.: Fascination of down scaling - Alice the sugar cube robot. *Journal of Micro-Mechatronics*, **1**(3), 177–189 (2002).

[9] Caprari, G., Siegwart, R.: Mobile micro-robots ready to use: Alice. *Proceedings of the IEEE IRS/RSJ International Conference on Intelligent Robots and Systems –IROS 2005*. Edmonton, Canada, 3845–3850 (2005).

[10] Dorigo, M.: *Optimization, Learning and Natural Algorithms*. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy (1992).

[11] Dorigo, M., Gambardella, L.M.: Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, **1**(1), 53–66 (1997).

[12] Eberhart, R.C., Shi, Y.: Particle swarm optimization: developments, applications and resources. *Proceedings of the 2001 Congress on Evolutionary Computation*, 81–86 (2001).

[13] Engelbretch, A.P.: *Fundamentals of Computational Swarm Intelligence*. John Wiley and Sons, Chichester, England (2005).

[14] Faigl, J., Krajnik, T., Chudoba, J., Preucil, L., Saska, M.: Low-cost embedded system for relative localization in robotic swarms. In: *Proc. of IEEE Int. Conf. on Robotics and Automation - ICRA'2013*, 993–998 (2013).

[15] Fister, S., Safaric, R., Fister Jr., I, Fister, I.: Parameter tuning of PI-controller with bat algorithm. *Informatica*, **40** 109–116 (2016).

[16] Fister, S., Fister Jr., I, Fister, I., Safaric, R.: Parameter tuning of PID controller with reactive nature-inspired algorithms. *Robotics and Autonomous Systems*, **84** 64–75 (2016).

[17] Garnier, S., Jost, C., Gautrais, J., Asadpour, M., Caprari, G., Jeanson, R., Grimal, A., Theraulaz, G.: The embodiment of cockroach aggregation behavior in a group of micro-robots. *Artificial Life*, **14**, 387–408 (2008).

[18] Gerkey, B.P. Vaughan, R.T., Howard, A.: The player/stage project: Tools for multirobot and distributed sensor systems. In: *Proceedings of the International Conference on Advanced Robotics*. Coimbra, Portugal, 317–323 (2003).

[19] Hassanien, A. E., Emary, E.: *Swarm Intelligence, Principles, Advances, and Applications*. CRC Press, Portland, USA (2015).

[20] Iglesias, A., Gálvez, A., Collantes, M.: Bat algorithm for curve parameterization in data fitting with polynomial Bézier curves. In: *Proc. of Cyberworlds 2015*, Visby (Sweden), IEEE Computer Society Press, Los Alamitos, CA, 107–114 (2015).

[21] Iglesias, A., Gálvez, A., Collantes, M.: Global-support rational curve method for data approximation with bat algorithm. In: *Proc. of Int. Conference Artificial Intelligence and Applications - AIAI'2015*, Bayonne (France). IFIP Advances in Information and Communication Technology, **458** 191–205 (2015).

[22] Iglesias, A., Gálvez, A., Collantes, M.: Iterative sequential bat algorithm for free-form rational Bezier surface reconstruction. *Int. J. Bio-Inspired Computation* (*in press*).

[23] Iglesias, A., Gálvez, A., Collantes, M.: Multilayer embedded bat algorithm for B-spline curve reconstruction. *Integrated Computer-Aided Engineering*, **24**(4), 385–399 (2016).

[24] Jackson, J.: Microsoft robotics studio: a technical introduction. *IEEE Robotics and Automation Magazine*, **14**(4) 82–87 (2007).

[25] Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: Proc. IEEE International Conference on Neural Networks, Perth, Australia. IEEE Computer Society Press, Los Alamitos, CA, 1942–1948 (1995).

[26] Kernbach, S., Thenius, R., Kernbach, O., Schmickl, T.: Re-embodiment of honeybee aggregation behavior in an artificial micro-robotic system. *Adaptive Behavior*, **17**(3), 237–259 (2009).

[27] Kennedy, J., Eberhart, R.C., Shi, Y.: *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco (2001).

[28] Kerth, G.: Causes and consequences of sociality in bats. *BioScience*, **58**(8), 737–746 (2008).

[29] JMcLurkin, J.: Stupid robot tricks: a behavior-based distributed algorithm library for programming swarms of robots. M.S. thesis, *Massachusetts Institute of Technology* (2004).

[30] McLurkin, J., Smith, J.: Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots. In: *Distributed Autonomous Robotic Systems*, **6**, Springer, 399–408 (2007)

[31] McLurkin, J., Lynch, A. Rixner, S. Barr, T. Chou, A.,. Foster, K., Bilstein, S.: A low-cost multi-robot system for research, teaching, and outreach. in: *Distributed Autonomous Robotic Systems*, *83*, Springer, 597–609 (2013).

[32] McLurkin, J., Smith, J. Frankel, J., Sotkowitz, D., Blau, D., Schmidt, B.: Speaking swarmish: Human-robot interface design for large swarms of autonomous mobile robots. In: *AAAI spring symposium*, 72–75 (2006).

[33] Michel, O.: Webots: professional mobile robot simulation. *Journal of Advanced Robotics Systems*, **1**(1) 39–42 (2004).

[34] Mondada, F. Franzi, E., Guignard, A.: The development of Khepera. *Proceedings of the 1st International Khepera Workshop*, HNI-Verlagsschriftenreihe, Heinz Nixdorf Institut, **64**, 7–14 (1999).

[35] Mondada, F., Gambardella, L.M., Floreano, D., Nolfi, S., Deneubourg, J.-L., Dorigo, M.: The cooperation of swarm-bots: physical interactions in collective robotics. *IEEE Robotics and Automation Magazine*, **12**(2), 21–28 (2005).

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

[37] Pugh, J., Raemy, X., Favre, C., Falconi, R., Martinoli, A.: A fast onboard relative positioning module for multirobot systems. *IEEE/ASME Transactions on Mechatronics*, **14**(2), 151–162 (2009).

[38] Ramezani, A., Chung, S.J., Hutchinson, S.: A biomimetic robotic platform to study flight specializations of bats. *Science Robotics*, **2**(3). Art. No. eaal2505, (2017).

[39] Rubenstein, M., Ahler, C., Nagpal, R.: Kilobot: A low cost scalable robot system for collective behaviors. In: *IEEE International Conference on Robotics and Automation – ICRA'2012*, 3293–3298 (2012).

[40] Rubenstein, M., Cabrera, A., Werfel, J., Habibi, G., McLurkin, J., Nagpa, N.l: Collective transport of complex objects by simple robots: theory and experiments. *International conference on Autonomous Agents and Multi-Agent Systems – AAMAS 2013*: 47–54 (2013).

[41] Rubenstein, M., Ahler, C., Hoff, N., Cabrera, A., Nagpal, N.: Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems*, **62**(7), 966–975 (2014)

[42] Rubenstein, M., Cornejo, A., Nagpal, N.: Programmable self-assembly in a thousand-robot swarm. *Science*, **345**(6198), 795–799 (2014).

[43] Gauci, M., Ortiz, M.E., Rubenstein, M., Nagpa, N.l: Error Cascades in Collective Behavior: A Case Study of the Gradient Algorithm on 1000 Physical Agents. *International conference on Autonomous Agents and Multi-Agent Systems – AAMAS 2017*, 1404–1412 (2017).

[44] Sahin, E.: Swarm robotics: from sources of inspiration to domains of application. In: Swarm robotics. *Lecture Notes in Computer Science*, **3342**, 10–20 (2005).

[45] Sauter, J. A., Matthews, R., Parunak, H. V. D., Brueckner, S. A.: Effectiveness of digital pheromones controlling swarming vehicles in military scenarios. *Journal of Aerospace Computing, Information, and Communication*, **4**(5) 753–769 (2007).

[46] Saska, M., Vonasek, V., Krajnik, T., Preucil, L.: Coordination and navigation of heterogeneous MAVUGV formations localized by a hawk-eye-like approach under a model predictive control scheme. *International Journal of Robotics Research*, **33**(10) 1393–1412 (2014).

[47] Schmickl, T., Thenius, R., Moeslinger, C., Radspieler, G,. Kernbach, S., Szymanski, M., Crailsheim, K.: Get in touch: cooperative decision making based on robot-to-robot collisions. *Autonomous Agents and Multi-Agent Systems*, **18**(1), 133–155 (2009).

[48] Seyfried, J., Szymanski, M., Bender, N., Estana, R., Thiel, M., Wörn, H.: The i-swarm project: intelligent small world autonomous robots for micro-manipulation. In: Swarm robotics. *Lecture Notes in Computer Science*, **3342**, 70–83 (2005).

[49] Suarez, P., Iglesias, A.: Bat algorithm for coordinated exploration in swarm robotics. In: Del Ser, J. (ed.): *Int. Conf. on Harmony Search Algorithm – ICHSA 2017*. Advances in Intelligent Systems and Computing, **514**. Springer, Singapore 134–144 (2017).

[50] Tan, Y., Zheng, Z.Y.: Research advance in swarm robotics. *Defence Technology Journal*, **9**(1) 18–39 (2013).

[51] Turgut A.E., Celikkanat, H., Gokce, F., Sahin E.: Self-organized flocking with a mobile robot swarm. *Int. Conf. on Autonomous Agents and Multiagent Systems – AAMAS 2008*, 39–46 (2008).

[52] Turgut A.E., Celikkanat, H., Gokce, F., Sahin E.: Self-organized flocking in mobile robot swarms. *Swarm Intelligence*, **2**(2) 97–120 (2008).

[53] Valdastri, P. Corradi, P., Menciassi, A., Schmickl,T., Crailsheim, K., Seyfried, J., Dario, P.: Micromanipulation, communication and swarm intelligence issues in a swarm microrobotic platform. *Robotics and Autonomous Systems*, **54**(10) 789–804 (2006).

[54] Vasarhelyi, G., Virgh, C., Tarcai, N., Somorjai, G., Vicsek, T.: Outdoor flocking and formation flight with autonomous aerial robots. In: *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems - IROS 2014*, 3866–3873 (2014).

[55] Vaughan, R.: Massively multi-robot simulation in Stage. *Swarm Intelligence*, **2**(2-4), 189–208 (2008).

[56] Wagner, I., Bruckstein, A.: Special Issue on Ant Robotics. *Annals of Mathematics and Artificial Intelligence*, **31**(1-4) (2001).

[57] Yang, X.-S.: *Nature-Inspired Metaheuristic Algorithms (2nd. Edition)*. Luniver Press, Frome, UK (2010).

[58] Yang, X.S.: A new metaheuristic bat-inspired algorithm. *Studies in Computational Intelligence*, Springer Berlin, **284**, 65–74 (2010).

[59] Yang, X. S..: Bat algorithm for multiobjective optimization. *Int. J. Bio-Inspired Computation*, **3**(5), 267-274 (2011).

[60] Yang, X.S., Gandomi, A.H.: Bat algorithm: a novel approach for global engineering optimization. *Engineering Computations*, **29**(5), 464–483 (2012).

[61] Yang, X.S.: Bat algorithm: literature review and applications. *Int. J. Bio-Inspired Computation*, **5**(3), 141–149 (2013).

[62] Yang, X.S.: *Nature-Inspired Optimization Algorithms*. Elsevier (2014).

[63] Yang, X.S.: *Nature-Inspired Computation in Engineering*. Studies in Computational Intelligence Series, **637**. Springer Switzerland (2016).

[64] Zahugi, E.M.H., Shabani, A.M., Prasad, T.V.: Libot: Design of a low cost mobile robot for outdoor swarm robotics. In: *Proc. of IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems - CYBER'2012*, 342–347 (2012)