Interplay of Two Bat Algorithm Robotic Swarms in Non-Cooperative Target Point Search

Patricia Suárez[1], Akemi Gálvez[1,2], Iztok Fister[3], Iztok Fister Jr.[3], Eneko Osaba[4], Javier Del Ser[4,5,6], Andrés Iglesias[1,2,†]

[1]University of Cantabria, Avenida de los Castros s/n, 39005, Santander, Spain
[2]Toho University, 2-2-1 Miyama, 274-8510, Funabashi, Japan
[3]University of Maribor, Smetanova, Maribor, Slovenia
[4]TECNALIA, Derio, Spain
[5]University of the Basque Country (UPV/EHU), Bilbao, Spain
[6]Basque Center for Applied Mathematics (BCAM), Bilbao, Spain
[†]Corresponding Author: `iglesias@unican.es`
`http://personales.unican.es/iglesias`

Abstract. In this paper, we analyze the interplay of two robotic swarms applied to solve a target point search in a non-cooperative mode. In particular, we consider the case of two identical robotic swarms deployed within the same environment to perform dynamic exploration seeking for two different unknown target points. It is assumed that the environment is unknown and completely dark, so no vision sensors can be used. Our work is based on a robotic swarm approach recently reported in the literature. In that approach, the robotic units are driven by a popular swarm intelligence technique called bat algorithm. This technique is based on echolocation with ultrasounds, so it is particularly well suited for our problem. The paper discusses the main findings of our computational experiments through three illustrative videos of single executions.

Keywords: swarm intelligence · swarm robotics · bat algorithm.

1 Introduction

A recent trend in swarm robotics (SR) is the use of multiple robotic swarms operating simultaneously within the same environment [1]. The most common case in the literature and real-life applications is the *cooperative mode*, defined by the cooperation among swarms to achieve a common goal. In contrast, little attention has been given so far in the literature to the *non-cooperative case*, where each swarm tries to solve its own goals with little (or none at all) consideration to any other factor external to the swarm (e.g., other swarms' goals). Note however that this non-cooperative case also arises in some practical applications. Aimed at filling this gap, in this work we focus on the behavioral pattern of robotic swarms under this non-cooperative regime.

In particular, in this work we consider the case of two robotic swarms $\mathcal{S}_1$ and $\mathcal{S}_2$ comprised by a set of μ and ν robotic units $\mathcal{S}_1 = \{r_1^i\}_{i=1,\dots,\mu}$, $\mathcal{S}_2 =$

$\{r_2^j\}_{j=1,\dots,\nu}$, respectively. For simplicity, we can assume that $\mu = \nu$ and that all robotic units are identical, i.e., $r_1^i = r_2^j$, $\forall i, j$. These robotic units are deployed within the same environment $\mathbf{\Omega} \subset \mathbb{R}^3$ to perform dynamic exploration. We assume that the geometry of $\mathbf{\Omega}$ is unknown to the robots and completely dark, so any vision sensor becomes useless to the robots. The goal of each swarm $\mathcal{S}_k$ is find a static target point $\mathbf{\Phi}_k$ $(k = 1, 2)$, placed in a certain (unknown) location of the environment. We assume that $||\mathbf{\Phi}_1 - \mathbf{\Phi}_2|| > \delta$ for a certain threshold value $\delta > 0$, meaning that both target points are not very close to each other so as to broaden the spectrum of possible interactions between the swarms. Although the environment is a 3D world, we consider the case of mobile walking robots moving on a two-dimensional map $\mathcal{M} = \mathbf{\Omega}|_{z=0}$.

Regarding the artificial intelligence method, there are several possibilities to build a robotic swarm under these constraints [1]. In this paper, we focus on a powerful bio-inspired swarm intelligence approach called *bat algorithm.* This choice is very natural as this algorithm is based on the echolocation behavior of some species of microbats living in dark environments such as caves, thus meeting our previous assumptions. This algorithm has shown to be very effective to address difficult continuous optimization problems involving a large number of variables [2, 3, 7], including the efficient navigation in dynamic indoor environments [4, 5]. A recent paper describes a physical and computational implementation of a swarm of bat algorithm-based robotic units. Such implementation is fully specialized in the sense that all components of the robots are designed and fully optimized to replicate the most relevant features of the real microbats and the bat algorithm as faithfully as possible [6]. This work is based on that approach. In fact, the physical robots, shown in Figure 1, are already built according to the implementation described in [6]. However, since in this paper we are interested in the computational features, our description will be based on computer simulations exclusively. To this aim, the physical robots are replaced by their digital models, shown in Fig. 2.

The structure of this paper is as follows: in Sect. 2 we provide a gentle overview about the bat algorithm. Our SR approach for non-cooperative target point search is described in Sect. 3 and illustrated using two independent robotic swarms in Sect. 4. The paper closes with the conclusions and some plans for future work in the field.

2 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 [7–9]. The algorithm is based on the echolocation behavior of microbats, which 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 [10]. The idealization of the echolocation of microbats is as follows:

1. Bats use echolocation to sense distance and distinguish between food, prey and background barriers.

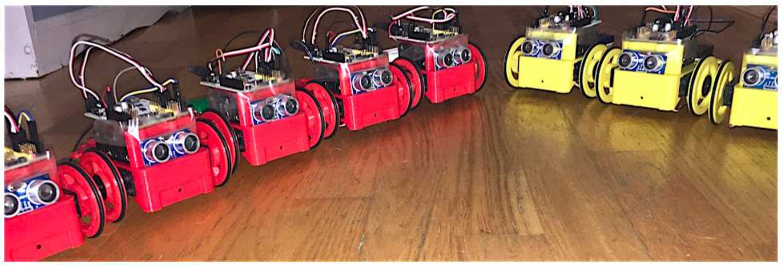

Fig. 1. Physical robots of the two robotic swarms in red and yellow, respectively.

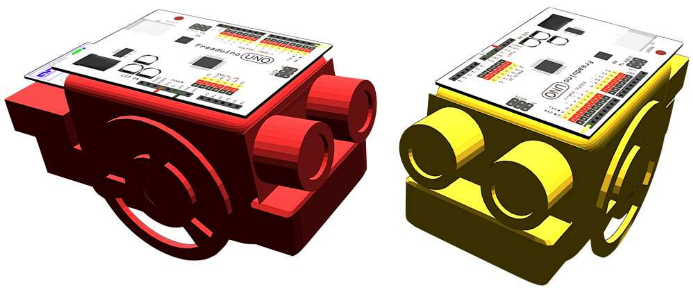

Fig. 2. Graphical models of the robots in Figure 1.

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}.

Some additional assumptions are advisable for further efficiency. For instance, 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. 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 $\mathbf{x}_i$

```
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 using eqns. (1)-(3)
 8:     if $\theta > r_i$ then
 9:       $\mathbf{s}^{best} \leftarrow \mathbf{s}^g$      //select the best current solution
10:       $\mathbf{ls}^{best} \leftarrow \mathbf{ls}^g$    //generate a local solution around $\mathbf{s}^{best}$
11:     end if
12:     Generate a new solution by local random walk
13:     if $\theta < \mathcal{A}_i$ and $\phi(\mathbf{x_i}) < \phi(\mathbf{x}^*)$ then
14:       Accept new solutions, increase $r_i$ and decrease $\mathcal{A}_i$
15:     end if
16:   end for
17:   $g \leftarrow g + 1$
18: end while
19: Rank the bats and find current best $\mathbf{x}^*$
20: return $\mathbf{x}^*$
```

Algorithm 1: Bat algorithm pseudocode

and velocity $\mathbf{v}_i$. The algorithm initializes these variables with random values within the search space. Then, the pulse frequency, pulse rate, and loudness are computed for each individual bat. Then, the swarm evolves in a discrete way over generations, like time instances until the maximum number of generations, $\mathcal{G}_{max}$, is reached. For each generation g and each bat, new frequency, location and velocity are computed according to the following evolution equations:

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

$$\mathbf{v}_i^g = \mathbf{v}_i^{g-1} + [\mathbf{x}_i^{g-1} - \mathbf{x}^*] f_i^g \tag{2}$$

$$\mathbf{x}_i^g = \mathbf{x}_i^{g-1} + \mathbf{v}_i^g \tag{3}$$

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. Then, search is intensified by a local random walk. 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: $\mathbf{x}_{new} = \mathbf{x}_{old} + \epsilon\mathcal{A}^g$,

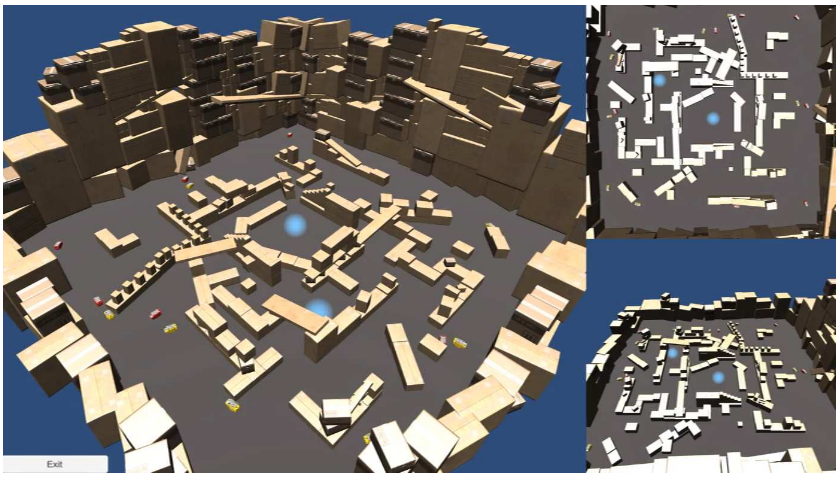

Fig. 3. Three different views of the graphical environment at initialization stage.

where ϵ is a uniform random number on $[-1, 1]$ 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 13-16). 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: $\mathcal{A}_i^{g+1} = \alpha\mathcal{A}_i^g$ and $r_i^{g+1} = r_i^0[1 - exp(-\gamma g)]$ where α and γ are constants. Note that for any $0 < \alpha < 1$ and any $\gamma > 0$ we have: $\mathcal{A}_i^g \to 0$, $r_i^g \to r_i^0$ as $g \to \infty$. Generally, each bat should have different values for loudness and pulse emission rate, which can be 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.

3 Bat Algorithm Method for the Robotic Swarms

In this paper we consider the synthetic closed environment shown in Figure 3. The figure is split into three parts for better visualization, corresponding to two side view cameras from different locations and a top view camera. The scene consists of a collection of cardboard boxes stacked in a messy way and forming challenging structures for the robots such as corridors, tunnels, dead ends, bifurcations, T-junctions, and the like. Although the robots of the two swarms are functionally identical, they are depicted in red and yellow color respectively for visualization purposes. Two blue spherical-shaped points of light mark the target points $\mathbf{\Phi}_k$ for the two robotic swarms. As explained above, the goal of each robotic swarm $\mathcal{S}_k$ is to find its corresponding target point $\mathbf{\Phi}_k$. In our approach, each robot moves autonomously, according to the current values of its fitness function and parameters, and communicates them to the other members of the swarm. To this aim, each virtual robot r^i_k is described at time instance j by a vector $\mathbf{\Xi}^{i,j}_k = \left\{ f^{i,j}_k, \mathbf{x}^{i,j}_k, \mathbf{v}^{i,j}_k \right\}$, where $f^{i,j}_k$, $\mathbf{x}^{i,j}_k = (x^{i,j}_k, y^{i,j}_k)$ and $\mathbf{v}_{i,j} = (v^{i,j}_{k,x}, v^{i,j}_{k,y})$ represent the fitness value, position, and velocity, respectively. The robots are deployed at initial random positions $\mathbf{x}^{i,0}_k$ and with random velocities $\mathbf{v}^{i,0}_k$ provided that they are restricted to move within the map $\mathcal{M}$. However, opposed to the usual procedure in swarm intelligence methods, we refrain from deploying the robots randomly within all the search space to avoid that some robots could accidentally initialize very near to the target, thus reducing the complexity of the problem. For instance, as shown in Fig. 3, the robots are initialized at random positions in the outermost parts of the map. For the robots motion, we assume that $\mathcal{M}$ is described by a tessellation of convex polygons $\mathcal{T}_{\mathcal{M}}$. Then, we consider the set $\mathcal{N}_{\mathcal{M}} \subset \mathcal{T}_{\mathcal{M}}$ (called the *navigation mesh*) comprised by all polygons that are fully traversable by the robots. At time j the fitness function $f^{i,j}_k$ can be defined as the distance between the current position $\mathbf{x}^{i,j}_k$ and the target point $\mathbf{\Phi}_k$, measured *on* $\mathcal{N}_{\mathcal{M}}$ as $f^{i,j}_k = ||\mathbf{x}^{i,j}_k - \mathbf{\Phi}_k||_{\mathcal{N}_{\mathcal{M}}}$ so our problem consists of minimizing the value of $f^{i,j}_k$, $\forall i, j, k$. This minimization problem is solved through the bat algorithm described in Sect. 2.

A critical issue when working with swarm intelligence techniques is the parameter tuning, which is well-known to be problem-dependent. Our choice has been fully empirical. For computational efficiency, we set the population size to 9 robots for each swarm, as larger values increase the number of collisions among robots and, hence, the computational time. The initial and minimum loudness and parameter α are set to 0.5, 0, and 0.6, respectively. We also set the initial pulse rate and parameter γ to 0.5 and 0.4, respectively. However, our results do not change significantly when varying these values slightly. All executions are performed until all robots reach the target point, no matter the number of iterations needed for completion. Our method has been implemented in *Unity 5* on a 3.8GHz quad-core Intel Core i5, with 16GB of DDR3 memory, and a graphical card AMD RX580 with 8GB VRAM. All programming code in this paper has been created in *JavaScript* using the *Visual Studio* programming framework.

4 Experimental Results

Our approach has been tested for many random initial locations of the robots in the scene. Only three of them is described here because of limitations of space, corresponding to three videos submitted as accompanying material. As above mentioned, the robots are initialized at the outermost parts of the scene, where the red and yellow swarms are randomly intertwined. Then, the bat algorithm starts and both swarms try to reach their corresponding target points, located in the middle of the central square of the scene for the yellow swarm and in a corner of the first ring around that square for the red swarm. The movement of the robots is driven by the bat algorithm but also affected by three factors: the complex and irregular geometry of the scene, the collisions with robots of the other swarm and their own, and the fact that the map is unknown to the robots. The later fact is clearly visible in *Video 1* (seconds[1] 14–16), when a yellow robot crosses the central square on one side without realizing that is near to the target point. However, at seconds 20–22 two other yellow robots reach the central square just at the center, so they are actually very near to the target. At that point, one of them becomes the current best, attracting the other members of the swarm over the iterations. Note also that, owing to the echolocation, the robots can detect static obstacles and other robots before they collide, thus avoiding crashing. In those cases, the robots move rapidly wandering to the left and right, occasionally multiple times, trying to overcome the obstacle (see, for instance, the red robots in bottom corner of main window at seconds 5–9). This also explains why the robots are constantly moving, even after reaching the target point, as they are trying to avoid colliding with other members of the swarm. In some cases this leads to very crowded formations such as the contact-less scrummage-like in seconds 16–19, where several robots from both swarms gathering at a specific location try to avoid multiple collisions simultaneously.

We remark the ability of the robots to avoid the collisions and move forward towards the target, showing the good performance of the bat algorithm for this task. This fact becomes even more evident in *Video 2*, where all yellow robots get trapped by the red robots in a dead end in the neighborhood of the target point of the red swarm for seconds 28–45, until one of the yellow robots eventually finds a way to escape, dragging the rest of the swarm in its movement towards its own target point. The yellow robots are leaving the dead end one by one until the last yellow robot, still trapped by five red robots (seconds 68–79) is able to circumvent the red swarm and leaves the area. At their turn, these five red robots find troubles to reach the target point (even when the other four members are already there) due to their particular orientation facing each other, thus preventing them from moving forward for a while. Finally, *Video 3* shows one of the rare examples of the robotic swarms able to reach their target points with minimal interaction between the swarms.

5 Conclusions and Future Work

In this paper, we discuss some behavioral patterns found from the interplay of two robotic swarms coexisting in the same environment and following a non-cooperative approach when trying to reach their individual goals. To this aim, we consider two robotic swarms driven by the bat algorithm and moving in an unknown closed environment. As shown in the videos, the bat algorithm allows

[1] All times are measured since the execution starts, i.e., excluding initial credits.

the robotic swarms that got trapped to move away while simultaneously avoiding to collide with the other robots, finding their targets in reasonable time.

In addition to the previous results, we found many other behavioral patterns for the robotic swarms. For instance, a qualitatively different behavioral pattern can be seen for the yellow robot nearest to the target point of the red swarm in seconds 21–31 of *Video 1*, when the robot stops moving while waiting for the red robots to gather around their target point and free the path to this robot to join its own swarm. Other patterns that we observed include moving in a formation (usually led by the global best of the swarm), aggregation patterns for intensive exploration near the optima, bifurcation moving patterns for simultaneous exploration and obstacle avoidance, cooperation among robots to force a robot of the other team to move away, and ability to escape from U and V configurations such as dead ends, among others. A full analysis of all these behavioral patterns and replicating these experiments with the physical robotic swarms in real life for comparison are part of our plans for future work in the field.

Acknowledgements
Research supported by: project PDE-GIR of the EU Horizon 2020 research and innovation program, Marie Sklodowska-Curie grant agreement No 778035; project #TIN2017-89275-R of Agencia Estatal de Investigación and EU Funds FEDER (AEI/FEDER-UE); project #JU12, of SODERCAN and EU Funds FEDER (SODERCAN/FEDER-UE); project EMAITEK of Basque Government.

References

1. Brambilla, M., Ferrante, E., Birattari, M., Dorigo, M.: Swarm robotics: a review from the swarm engineering perspective, Swarm Intelligence, 1–41 (2013).
2. Iglesias, A., Gálvez, A., Collantes, M.: Multilayer embedded bat algorithm for B-spline curve reconstruction. Integ. Computer-Aided Eng., **24**(4), 385–399 (2017).
3. Iglesias, A., Gálvez, A., Collantes, M.: Iterative sequential bat algorithm for free-form rational Bézier surface reconstruction. Int. J. Bio-Inspired Computation, **11**(1), 1–15 (2018).
4. Suárez, P., Iglesias, A.: Bat algorithm for coordinated exploration in swarm robotics. Advances in Intelligent Systems and Computing, 514, 134–144 (2017).
5. Suárez, P., Gálvez, A., Iglesias, A.: Autonomous coordinated navigation of virtual swarm bots in dynamic indoor environments by bat algorithm. Int. Conf. in Swarm Intelligence, ICSI 2017. Lecture Notes in Computer Science, 10386, 176–184 (2017).
6. Suárez, P., Iglesias, A., Gálvez, A.: Make robots be bats: specializing robotic swarms to the bat algorithm. Swarm and Evolutionary Computation (*in press*) DOI: 10.1016/j.swevo.2018.01.005.
7. Yang, X.S.: A new metaheuristic bat-inspired algorithm. Studies in Computational Intelligence, Springer Berlin, 284, pp. 65–74 (2010).
8. Yang, X. S..: Bat algorithm for multiobjective optimization. Int. J. Bio-Inspired Computation, **3**(5), 267–274 (2011).
9. Yang, X.S., Gandomi, A.H.: Bat algorithm: a novel approach for global engineering optimization. Engineering Computations, **29**(5), 464–483 (2012).
10. Yang, X.S.: Bat algorithm: literature review and applications. Int. J. Bio-Inspired Computation, **5**(3), 141–149 (2013).