Journal of Computational Science 29 (2018) 107–118

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Control optimization of an aerial robotic swarm in a search task and its adaptation to different scenarios

Pablo Garcia-Aunon*, Antonio Barrientos Cruz

Centre for Automation and Robotics (CAR), Technical University of Madrid (UPM), C/ José Gutiérrez Abascal, 2, 28006 Madrid, Spain

ARTICLE INFO

Article history:
Received 1 August 2018
Received in revised form
12 September 2018
Accepted 9 October 2018
Available online 13 October 2018

Keywords:
Swarm robotics
Optimization
Modeling
Search

ABSTRACT

In many cases, the control system for robotic swarms are complex behavioral networks (frequently probabilistic finite state machines) with several parameters to be tuned. Their selection has a high impact on the performance of the swarm carrying out a given task. In the past, those parameters have been optimized using automatic methods, whereas in other cases the network is simplified and tuned making use of expert knowledge. The problem becomes trickier when the performance depends not only on these control parameters, but also on other variables that cannot be selected by the designer (such as the size of the scenario, or the number of available agents). Moreover, there usually exist factors that inject noise in the measured outcome, such as the initial conditions, making the task more difficult to be analyzed. This work proposes and compares principled methods that address these two issues: the optimal configuration of controls with a high dimensional configuration space, that at the same time must be optimized for a broad range of scenarios. As a testbed task, search on a rectangular area is studied. We show that our proposal successfully addresses this complex problem. Moreover, our approach may be also implemented to configure subtasks inside a global mission, or on single behaviors that must be configured on-line depending on external state values.

List of abbreviations

BIC	Bayesian Information Criterion
BLR	Bayesian Linear Regression
BOpt	Bayesian Optimization
FSM	Finite State Machine
GA	Genetic Algorithm
GP	Gaussian Process
PFSM	Probabilistic Finite State Machine
UAV	Unmanned Aerial Vehicle

1. Introduction

1.1. Swarm robotics: benefits and complexity

Collective behaviors are social processes and events, which do not reflect an existing social structure (laws, conventions, and institutions), but emerge in a spontaneous way. Although this definition was initially used to define human behaviors, it was soon clear that it may also be used for animals. Members of a group act in an ordered and structured way without having explicitly defined rules for specific purposes, and normally without a complex communication system between them. Swarm behaviors are a particular case of collective behavior, in which a large number of members are involved. For instance, swarm behavior involves ants and bees foraging processes, schools of fishes, flocks of birds (if the number of members is high) and crowd behaviors, among many others.

Also, the collective intelligence that emerges with the actions of these simple members of the swarm has been named as swarm intelligence. This is considered a type of artificial intelligence and allows to approach complex problems from a completely new perspective. Basically, swarm intelligence takes examples of swarms behaviors in nature and uses them as a metaphor to propose algorithms to solve mathematical problems, which cannot be solved using more traditional methods [1]. Particle Swarm Optimization, Ant Colony Optimization, Glowworm Swarm Optimization, Self-propelled Particles and Artificial Bee Colony Algorithm are some examples of algorithms and methods to solve mathematical problems by means of swarm intelligence.

* Corresponding author.
E-mail addresses: pablo.garcia.aunon@upm.es (P. Garcia-Aunon), antonio.barrientos@upm.es (A. Barrientos Cruz).

https://doi.org/10.1016/j.jocs.2018.10.004
1877-7503/

Some researches started bringing the ideas of swarms in nature to the robotics field. Goals are achieved using a high number of homogeneous robots, which behave following simple and mostly reactive rules. The pioneers of swarms robotics appeared in the late 80's and the beginning of the 90's, for instance [2–4]. Swarm robotic systems present some well known advantages such as robustness, simplicity, scalability and flexibility [5]. On the other hand, if we focus on the algorithm development for teams of robots made up by swarms, we face four main problems: (i) creation of behaviors at agent level and their interactions to properly accomplish a given task; (ii) crossing the reality gap between simulations and real experiments; (iii) difficulties to proper model the outcome of the swarm as a team; (iv) optimization of the algorithms at agent level. In this work we address the third and the fourth ones: modeling the swarm behavior as a team of robots and how to properly tune its algorithm in order to achieve better performances when carrying out the assigned task. These both challenges have been intensively studied in the past, and a good review of the automatic design methods can be found in [6].

1.2. Swarm modeling and optimization techniques

1.2.1. Modeling

Modeling is interesting from the engineering perspective because it allows to predict with some certainty the performance of the swarm. The idea is to mathematically represent some characteristics of the individuals (microscopic modeling) or of the group (macroscopic modeling), in order to predict specific outcomes of the group related to the achievements of the goals.

In the microscopic modeling, the interactions between individuals and between individuals and the environment are modeled. Inferring analytically a global behavior based on these models is usually infeasible and therefore, most of these models are tested in simulations. In [7] a microscopic model based on a probabilistic finite state machine (PFSM) is proposed for a stick-pulling swarm of robots. It is compared with another PFSM macroscopic model (using the Webots simulator) and with experimental results using Khepera robots. That work shows discrepancies between simulations and experiments due to the so-called reality gap, that is the inability to properly represent in simulations all the real world factors that have an impact on the task development. In a similar procedure, in [8] a connected swarm of robots is individually modeled with a PFSM, and macroscopically described by a different PFSM. Based on geometrical properties, some of the parameters of the models are assumed; these assumptions are validated through simulations. Again, there exist discrepancies between the model and the simulations because of the simplifications assumed in the model. However, thanks to this approach, valuable results can be obtained without expensive time-consuming simulations.

Trying to reduce the complexity of the macroscopic model, in [9] a microscopic model based on the Langevin equation is proposed, deriving in the Fokker-Planck equation that describes the complete group of robots. Thanks to this approach, the needed information to tune the model is reduced, saving time either in simulations and experiments with real robots. However, successfully inferring macroscopic effects from microscopic interactions is only possible with simple behavioral networks.

A very different approach was proposed in [10], where macroscopic variables are explicitly modeled by rate equations in a stick-pulling task. Those equations depend on parameters that are experimentally determined, getting finally how the macroscopic variables (such as the number of robots in search mode, or the number of unextracted sticks) change over time.

In [11] another PFSM is used to predict the performance of an adaptive control for a foraging task. Each agent adapts internal parameters over the task, modifying its transitions from one state to other. This adaptation poses an additional challenge for modeling the swarm. First, a FSM is proposed for each individual and transformed into a PFSM. The probabilities of the state transitions are afterwards estimated with experiments and analytical approaches, finally leading to the solving of the rate equations.

1.2.2. Optimization

Once we propose a control law for a swarm, there will be probably several parameters to be set. In many works where those algorithms are described, the selection of the values for these parameters have had little attention and normally have been set based on the intuition of the designer, or just by a simple trial and error approach. The more complex the task is, the more complex the control is likely to be, and therefore, the more parameters to be tuned there will potentially be. In those cases, trial and error approaches, or decisions based on expertise, become unfeasible options, and a more principled optimization method must be considered. In literature, this optimization procedure is normally included in the so-called automatic design process [6].

In most of these cases, there is not the chance of analytically optimizing the proposed problem. This situation is usual because we are dealing with multi-robot systems, with the consequent inherent complexity of modeling the interactions between agents, and between agents and environment. Note also that these problems naturally lead to very noisy outcomes (i.e. high variations of performance measurements), where it is complicated to identify proper inputs that describes them. In the swarm robotics field, and also in other areas, the genetic algorithms (GA, also referred as artificial evolution) have been widely used to search for suboptimal solutions. In the case here studied, it leads to an improvement of the individual behaviors of the agents, but it has been also implemented for other missions, such as task planning and allocation [12].

Within the Swarm-Bots project, in [13] three behaviors are improved using artificial evolution: movement synchronization, coordinated motion and coordinated motion avoiding holes placed on the test area. For each behavior, a neural controller is implemented, whose weights are the parameters to be optimized with different evolutionary techniques. More specifically for aerial swarms, in [14], a GA was used to set the parameters of 5 possible behaviors and 11 transitions for search and suppression of enemies missions. To coordinate the Unmanned Aerial Vehicles (UAVs), virtual pheromones are used, tracked by a central controller, which distributes the information among them. A total of 5 parameters were tuned with the GA, obtaining only a slight improvement compared with the initial population.

In [15], another GA is used to set the weights of a neural net, which controls a group of robots. The robots must explore a maze without getting stuck in fixed positions or hitting the walls. First, a suboptimal solution is found using a simulator, and afterwards it is tested in 8 real robots.

In [11] a foraging algorithm is optimized. 7 bounded parameters are set in order to obtain a suboptimal solution using as fitness function the net energy of the swarm. With a population of 30 members, creating an offspring of 15 members using linear crossover, the fitness function over the macroscopic model is evaluated for different environmental conditions and only the best members are preserved for the next generation. With this method, up to 93% of the rough ideal performance is reached.

Although the GAs have shown good performances in finding solutions, they also present important drawbacks. First, they are very intensive in computation, needing to analyze many members during several generations. This limits its direct application on real robots. Second, they must be tuned before starting the optimization; the number of members, the crossover method, the initial population or the mutation rate are only some of the decisions

Table 1
Summary of past works regarding optimization of robotic swarms.

Work	Task	Control	Number of parameters	Optimization	Adaptation to scenario
[13]	Synchronization Coordinated Motion Hole Avoidance	Neural Network	14 10 18	GA	No
[14]	Search and Suppression	PFSM	5	GA	No
[15]	Exploration	Neural Network	220	GA	No
[11]	Foraging	FSM	7	GA	No
[16]	Foraging and Aggregation	FSM Neural Network	6 50	AutoMoDe Evostick	No
[17]	Foraging	PFSM	7	GA	No
This Work	Search	Behavior network	23	GA	Yes

to be made, and most of the times are chosen only based on the experience of the designer.

AutoMoDe [16] is a framework that allows the automatic design of robotic swarms, considering a finite state machine as control. It reduces the reality gap problem, since it prevents overfitting during the design process, looking for a trade-off between the variance and the bias of the learned data set. AutoMoDe does not only create the FSM and tune the probability transition between states, but also the parameters inside them. AutoMoDe-Vanilla is an application example for aggregation and foraging tasks with a total of six behavior and six conditions available to be selected for the the FSM. With a racing algorithm as the optimization method, the results are compared with Evostick (another automatic design method for robotic swarms), showing that although AutoMoDe-Vanilla has lower representational power, it has more ability to reduce the reality gap when comparing simulations with experiments.

Finally, in [17], a foraging algorithm based on a PFSM is optimized with a GA, with a total of 7 variables to be tuned. In each of the 100 generations, 100 members are evaluated, each of which is run 8 times due to the noise in the fitness function. There is a stabilization in the fitness function around 20 generations, with improvements up to 4 times better than the best initial member. It is also important to point out that a wide variety of conditions in the simulations are considered. The solutions are also tested in real robots, showing that communication between them when using virtual pheromones is feasible.

In Table 1, the above mentioned works have been summarized indicating the task to be solved, the control algorithm, the number of parameters to be optimized, the optimization method, and whether the optimized parameters adapt to the scenario parameters.

A particular mention deserves the algorithms based on behaviors, developed in the 80's by Brooks [18] and Arking [19], among others. The behavior-based controls basically consist of trying to decompose the task in subtasks and design specific controls to solve them one by one, integrating them to construct more complex structures. Because of this, learning is easier since the subtasks are fundamentally simpler. This control approach has been applied to solve several tasks, such as formation control [20] and flocking [21], navigation [22] and motion of generic robotic systems [23].

In these cases, the optimization is carried out gradually, and techniques such as reinforcement learning may be used. However, they pose important drawbacks that cannot be neglected. On the one hand, it is mandatory that the complete task can be decomposed in subtasks, condition that it is not always easy or even possible. The more subtasks we can identify, the easier the learning at each step will potentially be, but the more design workload will be needed. On the other hand, the solution, and therefore the way of optimizing it, is completely aimed. This means that we leave few room for the system to find by its own solutions with better performances than the ones defined from the beginning.

1.3. Our proposal

Reviewing the state of art about robotic swarms modeling and optimization (i.e. a specific phase of the automatic design) one realizes that although there have been remarkable advances in these areas, there are general cases in which the approaches proposed in the past would be insufficient:

- *Performance depending on the scenario:* the performance of the task at hand, and therefore the optimal values of the control, depends on the scenario. This means that it depends on parameters not selectable nor tunable inside the control of the robots. One of these scenario parameters could be, for instance, the size of the area in a foraging task, or the number of agents.
- *Noisy tasks:* the performance of the task, freezing the scenario parameters, with the same control configuration, is noisy. It may be so because the process itself is stochastic, because it depends on the initial conditions, or because there are some variables that have not been considered as scenario parameters, and therefore, have not been frozen between different trials.
- *Complex behavioral networks:* in past works, the number of parameters (such as the transition probabilities in a PFSM or internal values of the behaviors) to be tuned have been usually less than 10 in the swarm robotics field, with some exceptions [16]. The more complex the network is, the more parameters to be tuned there will be.

In order to address these limitations, we make use of a behavioral network that comprises 6 behaviors with a high dimensional configuration space with 23 variables, that was originally published in [24]. These variables are optimized using two different methods: a Genetic Algorithm (GA) and a Bayesian Optimization (BOpt). Moreover, 5 more parameters that cannot be selected (scenario parameters) are considered, implying that the algorithm needs to adapt to them (the optimized values are function of these 5 parameters). Therefore, we propose two regression methods to configure the control for any scenario: a Bayesian Linear Regression (BLR) and a Gaussian Process (GP). Finally, both configuration models are compared with random configurations in terms of efficiency. Additionally, the efficiency is also modeled in order to predict it. In Fig. 1 an overview of this work has been presented, indicating the section number where each item appears.

As testbed we consider a very studied task: search of targets in a given area with Aerial Unmanned Vehicles (UAVs). Given that the UAVs provide clear advantages in these missions over other autonomous systems, they have been intensively studied and developed since their extensive use at the end of the 70s [25]. The search task itself might be extended to similar but more complex

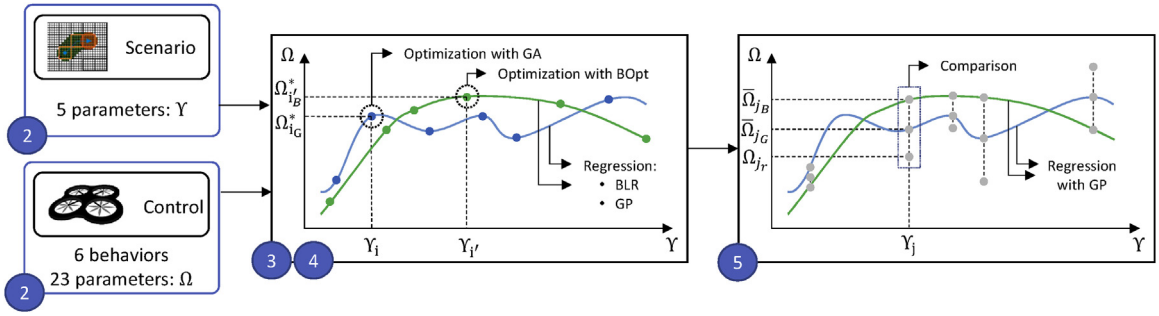


Fig. 1. Overview of the proposed work. On the lower left corner it is indicated the section in which each item is presented. In Section 2, the scenario and the control characteristics are introduced. In Sections 3 and 4 a data set of suboptimal configurations Ω_i^* is generated with two optimization methods, a Genetic Algorithm (GA, blue color) and a Bayesian Optimization (BOpt, green), depending on the scenario parameters Υ_i. Two regression methods are implemented and compared, a Bayesian Linear Regression (BLR) and a Gaussian Process (GP). In Section 5, the learned regressions are compared for new scenarios, considering also random configurations of the algorithm, Ω_r. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

tasks such as surveillance, exploration, patrolling, area coverage and the so-called travel salesman problem. There have been multiple approaches to solve this task, among which we can name the use of virtual pheromones [26,27], potential fields [28], flocking techniques [29], probabilistic maps [30],and heuristics [31–33].

This works presents an in-depth and principled method to analyze the noise of the mission outcome, the optimization of the algorithm presented in detail in [24], and how it adapts to different scenarios. In Section 2 the search problem is presented, defining the scenario and the agents, as well as the control algorithm. We also remark the complexity of the problem at hand, studying the noise due to the initial conditions; in Section 3 and 4 we present the regression and the optimization methods to be studied; their comparison is afterwards shown in Section 5, together with the final performance of the algorithm. Finally, in Section 6 the conclusions and future works are presented.

2. Problem formulation

In this Section, we state the problem at hand, briefly presenting the scenario parameters (i.e. variables of the problem that are externally imposed) and the control algorithm. Both scenario and algorithm are explained in detail in [24].

2.1. Search task, scenario and agents

The studied task consist of completely observing a given search space, which is rectangular with variable area A_s [m^2] and aspect ratio f_A. The space is subdivided in cells and the agents move only between the centers of those cells.

The search is performed by a team of N_a multicopters, capable of flying at a given altitude and commanded velocity v_n. In order to detect the targets in the search area, the agents are equipped with a downward-looking camera whose footprint is a circular area of radius R_f. As any other standard multicopter, the considered agents are holonomic and are able to fly in any direction. The available energy is limited, and the consumption depends on the flied distance and on the changes in the flying direction.

Table 2
List of parameters that define the scenario and their range of values considered.

Parameter	Description	Range of values
A_s/N_a	Search area per agent	[2000,15,000] m^2
N_a	Number of agents	[2,30]
v_n	Nominal velocity of the agents	[2,20] m/s
R_f	Radius of the sensor footprint	[5,20] m
f_A	Aspect ratio of the search area	[0.25,1]

These 5 parameters constitute the so-called scenario parameters, that is variables that cannot be chosen by the designer, but depend on the given area where the search must be conducted, and on the agent characteristics. In this work and in the original paper the tuple of scenario parameters is named with Υ. In Table 2, the list of the scenario parameters has been presented, together with their range of values considered.

In order to measure the performance of the algorithm for the studied task, we define the efficiency as the minimum theoretical time needed to observe the area, divided by the real time needed (t_n):

$$E = \frac{t_i}{t_n} \tag{1}$$

where $E \in (0, 1)$, and t_i is the ideal or minimum time needed:

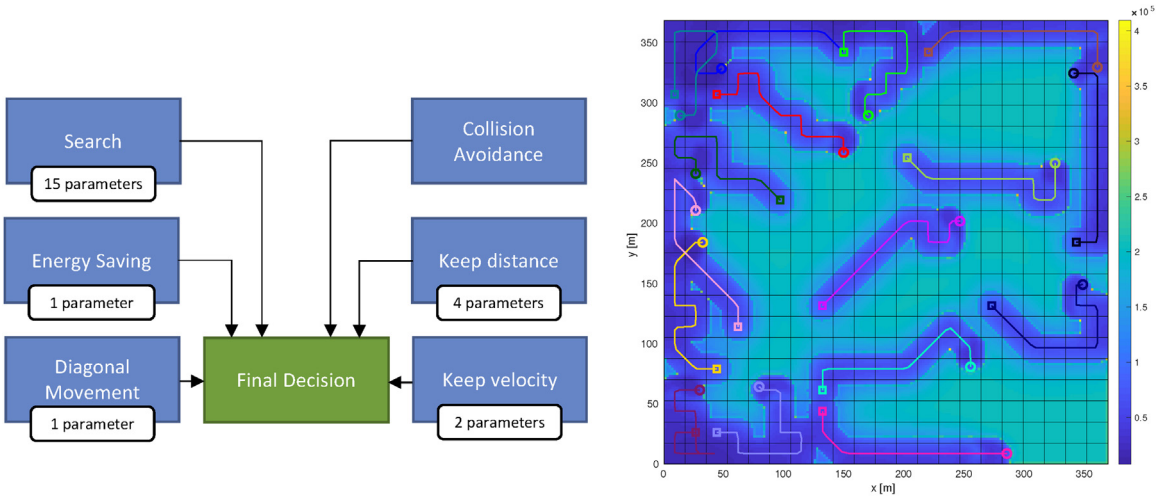
$$t_i = \frac{A_o - A_i}{2R_f v_n N_a} \tag{2}$$

being A_i the initial observed area, and A_o the actual observed area during the mission. Note that in some missions, depending on the search area per agent, the footprint sensor radius and the configuration of the algorithm, it may be $A_o < A_s$ due to the limited amount of available energy.

2.2. Control algorithm

In order to control the movements of the agents, a network composed by 6 behaviors is implemented. The algorithm was originally presented in detail in [24], and its performance compared with other heuristic algorithms for different search scenarios (plain, with obstacles, with target probability distribution, and with obstacles and target probability distribution). Each behavior counts with parameters that must be tuned, on which the overall performance of the algorithm depends. The suboptimal values of those parameters depend on the scenario, defined by the variables depicted in Table 2.

It must be remarked that the complete process of optimization exposed in this work was carried out for plain scenarios. However, as we showed in the previous work, it can be straightforwardly adapted to work in the other scenarios (i.e. with obstacles and/or with prior knowledge of the possible position of the targets), reaching good performances. In Fig. 2(b) an example of a search task is shown. Each agent and its path has been drawn with a different color; the squares indicate the initial position of the agents, whereas the circles indicate their current positions.



(a) Behaviors set (b) Search example

Fig. 2. In (a), a schematic view of the behaviors set that controls the movements of the agents, indicating the number of parameters to be tuned. The algorithm counts with 23 parameters. In (b), an example of the search task with 16 agents. The color bar represents the L1 layer of pheromones. Note that the area is divided in cells, and the agents move between their centers. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

2.3. Complexity of the problem

As already seen, the algorithm is made up by 6 behavior, each of which contains parameters to be set. Besides these 23 parameters, there exist another 5 variables that define the scenario, see Table 2. Each scenario is defined by a tuple of parameters $\Upsilon = \{A_s/N_a, N_a, v_n, R_f, f_A\}$; instead of using Υ, we will adimensionalize them using their ranges of values, so that every element i is $\tilde{\Upsilon}_i \in [0, 1]$. For each scenario $\tilde{\Upsilon}$, there may exist an optimal configuration Ω^* of those 23 parameters of the algorithm so that the efficiency function E, see Eq. (1), is maximized:

$$\Omega^*\left(\tilde{\Upsilon}\right) = \underset{\Omega}{\operatorname{argmax}}\, E\left(\tilde{\Upsilon}, \Omega\right) \tag{3}$$

Let us suppose that we want to find Ω^* by just sweeping its configuration space for a specific scenario. Dividing the space in a uniform grid is not a feasible option, since there are 23 variables; if we wanted for instance 10 points per variable, we would have 10^{23} possible configurations.

2.4. Efficiency noise due to the initial conditions

If we freeze the scenario parameters $\tilde{\Upsilon}$ and the configuration Ω, the efficiency varies if the initial conditions change. This means that among different simulations (or trials), the efficiency measured is noisy. In order to analyze it, we run 500 simulations for 10 different scenarios. For each scenario, we set the variables of the algorithm with 10 random configurations. A total of 50,000 simulations have been therefore carried out, which is a high-enough number of experiments to consider them as representative. For each scenario i and for each configuration j, we normalize the efficiency by subtracting its mean, and dividing it by the standard deviation of the samples:

$$\tilde{E}_{i,j,n} = \frac{E_{i,j,n} - \hat{E}_{i,j}}{\hat{\sigma}_{i,j}} \tag{4}$$

where $\hat{E}_{i_{i,j}}$ and $\hat{\sigma}_{i,j}$ are the estimated mean and standard deviation values respectively, and $n = 1, \ldots, 500$ is the trial number. The normalized efficiencies $\tilde{E}_{i,j,n}$ can now be compared among each scenario and each configuration. In Fig. 3 the histogram of the normalized efficiencies has been plotted. Using the Bayesian Information Criterion (BIC), we look for the best probability distribution to fit the efficiency, 4 of which have been plotted with the histogram.

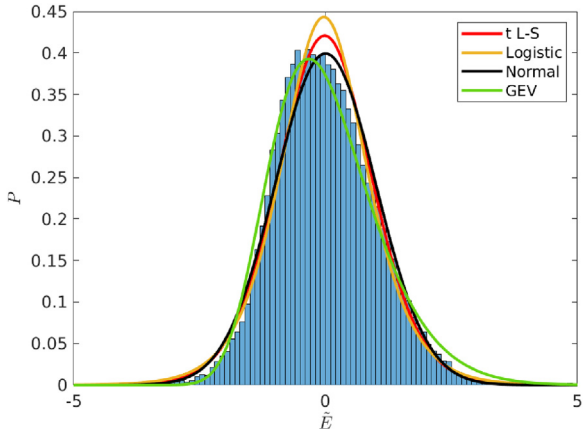


Fig. 3. Normalized efficiency of 10 different scenarios, with 10 random configurations for each of them. Each simulation has been repeated 500 times. The best 4 probability distributions have been shown.

Table 3
Bayesian Information Criterion (BIC) for different distributions to fit the noise of the efficiency due to the variation of the initial conditions, ordered in ascending values. t L-S, t Location Scale; GEV, Generalized Extreme Value; EV, Extreme Value; GP, Generalized Pareto.

	t L-S	Logistic	Normal	GEV	EV	Rayleigh	GP
BIC ($\cdot 10^{-5}$)	1.413	1.418	1.418	1.425	1.612	(0.944,1.654)	2.452

In Table 3 the BIC vales have been shown for each of the 7 distributions fitted. As it can be seen, there are not big differences between the first 4 ones. The Normal (or Gaussian) distribution present some beneficial characteristics that ease its implementation when using Bayesian approaches and therefore, it will be assumed that the efficiency values are drawn from a normal distribution:

$$E \sim \mathcal{N}\left(\eta_E, \lambda_E^{-1}\right) \tag{5}$$

To reduce the number of simulations needed to accurately identify the mean η_E and the variance of the distribution $\sigma_E^2 = \lambda_E^{-1}$, a Bayesian approach may be applied. Putting a normal and a gamma prior distribution over them:

$$\eta_E \sim \mathcal{N}\left(\eta_0, (\kappa_0\lambda_E)^{-1}\right) \tag{6a}$$

$$\lambda_E \sim \text{Ga}\left(\alpha_0, \beta_0\right) \tag{6b}$$

and using the Bayes rule, we get the following posterior marginal distribution [34]:

$$\lambda_E \sim \text{Ga}\left(\alpha_n, \beta_n\right) \tag{7a}$$

$$\eta_E \sim \text{T}_{2\alpha_n}\left(\eta_n, \frac{\beta_n}{\alpha_n\kappa_n}\right) \tag{7b}$$

$$\eta_n = \frac{\kappa_0\eta_0 + n\hat{E}}{\kappa_0 + n} \tag{7c}$$

$$\kappa_n = \kappa_0 + n \tag{7d}$$

$$\alpha_n = \alpha_0 + n/2 \tag{7e}$$

$$\beta_n = \beta_0 + 1/2\sum_{n=1}^{N}\left(E_n - \hat{E}\right)^2 + \frac{\kappa_0 n\left(\hat{E} - \eta_0\right)^2}{2\left(\kappa_0 + n\right)} \tag{7f}$$

where Ga is the Gamma distribution and $\text{T}_{2\alpha_n}$ is the t-Student distribution. Having assumed prior distributions over the mean and the precision of the efficiency, the posteriors are calculated sequentially, eliminating the overfitting effect due to the low number of samples. For each scenario, a maximum of 40 trials will be simulated, but the simulations will be stopped earlier if the confidence over the distribution is good enough. This way, we reduce the number of needed simulations, increase the accuracy of the result and have a feedback about the certainty of the fitting of the data to the model.

We have assumed that a given configuration in a specific scenario reaches an efficiency distribution, which is Gaussian, with a mean value of η_E and a standard deviation σ_E. We can then compare two configurations for a common scenario. Given two random and independent variables (efficiencies) E_1 and E_2 drawn from two Gaussian distributions $\mathcal{N}\left(\eta_{E_1}, \sigma_{E_1}^2\right)$ and $\mathcal{N}\left(\eta_{E_2}, \sigma_{E_2}^2\right)$ respectively, the probability that E_1 is greater than E_2 is represented by:

$$\begin{aligned} P(E_1 > E_2) &= P(E_1 - E_2 > 0) \\ &= 1 - P(E_1 - E_2 \leq 0) \end{aligned} \tag{8}$$

A new variable $E_{1,2} = E_1 - E_2$ is defined, which is also drawn from a normal distribution. Its mean $\eta_{E_{1,2}}$ and standard deviation $\sigma_{E_{1,2}}$ are given by:

$$\eta_{E_{1,2}} := E(E_{1,2}) = \eta_{E_1} - \eta_{E_2} \tag{9a}$$

$$\sigma_{E_{1,2}}^2 := Var(E_{1,2}) = \sigma_{E_1}^2 - \sigma_{E_2}^2 \tag{9b}$$

where $E()$ indicates expectation and $Var()$ variance. Finally, the probability that $E_{1,2}$ is positive is calculated as:

$$\begin{aligned} P(E_{1,2} > 0) &= 1 - P(E_{1,2} \leq 0) \\ &= 1 - P\left(\frac{E_{1,2} - \eta_{E_{1,2}}}{\sigma_{E_{1,2}}} \leq \frac{-\eta_{E_{1,2}}}{\sigma_{E_{1,2}}}\right) \\ &= 1 - \Phi\left(\frac{-\eta_{E_{1,2}}}{\sigma_{E_{1,2}}}\right) \end{aligned} \tag{10}$$

being Φ the cumulative distribution function of a normal distribution with mean equal to 0 and variance equal to 1.

2.5. Fitness function

In Section 2.1 we have assumed that each agent accounts with a specific amount of available energy; once it is consumed, it stops flying and therefore, it stops observing the scenario. Depending on how efficient the configuration of the algorithm is, and on the scenario characteristics, it may happen that the search area is not completely observed. In those cases, the efficiency will be penalized by only considering the 25% of the efficiency. Note that it may happen that for the same scenario, with the same configuration of the algorithm, the task may be completed or not depending on the initial conditions. In those cases, the efficiency penalization will be only applied for the trials in which the search has not been completed.

As we have seen, the efficiency depends on the initial conditions, for a given configuration Ω and scenario Υ. We consider that a configuration that achieves an efficiency with a small standard deviation can be considered as a robust configuration, since it is capable of carrying out the task within similar periods of time, independently from the initial conditions. Therefore, instead of evaluating the goodness of the configuration with the efficiency, we define a fitness function that accounts this:

$$f = E \cdot \beta_f \cdot \beta_e \tag{11a}$$

$$\beta_f = \frac{1}{3}\left(2 + \frac{\tanh\left(50(\sigma_c - \sigma)\right) + 1}{\tanh\left(50\sigma_c\right) + 1}\right) \tag{11b}$$

$$\beta_e = \begin{cases} 1 & \text{if search completed} \\ 0.25 & \text{if search not completed} \end{cases} \tag{11c}$$

where $\beta_f \in [2/3, 1]$ is a correction factor that penalizes high standard deviations, being $\beta_f(\sigma = 0) = 1$; β_e is the penalization in case the search is not completed. Based on experience, it has been chosen $\sigma_c = E/10$, considering this value as reachable if the algorithm is properly configured.

3. Regression methods

Once we have generated a big-enough data set with N suboptimal configurations Ω_i^* corresponding to N scenarios Υ_i, we can model them, and through a regression we are able to predict suboptimal configurations for new scenarios. This is then a supervised learning problem and two methods are studied: a Bayesian Linear Regression and a Gaussian Process.

Each suboptimal configuration is computationally expensive, as it will be shown in Section 4, and therefore, it is unfeasible to obtain a large data set. Then, it is essential that the regression method used to model them prevent the overfitting phenomena. The Bayesian Linear Regression and the Gaussian Process both fulfill this requirement. At the same time, they also give information about the noise and uncertainty of predictions [35].

3.1. Bayesian Linear Regression

The first model we propose is a Bayesian Linear Regression (BLR) for each of the 23 parameters of the algorithm. Given a target value $t_{i,n}$, variable number i of Ω_n^*, corresponding to the scenario number n, and $\mathbf{x}_n$, the inputs of the model for that scenario Υ_n, we assume:

$$t_{i,n} = y_{i,n}(\mathbf{x}_n, \mathbf{w}_i) + \epsilon_{i,n} \tag{12}$$

being $\epsilon_{i,n}$ a Gaussian noise, and the underlying functions $y_{i,n}(\mathbf{x}_n, \mathbf{w}_i)$ linear:

$$y_{i,n}(\mathbf{x}_n, \mathbf{w}_i) = \sum_{h=0}^{H}\sum_{m=0}^{M} w_{h+m,i} x_{h,n}^m = \mathbf{w}_i^T \phi(\mathbf{x}_n) \tag{13}$$

where H is the size of the input vector, M is the order of the polynomial, $w_{0,i}$ is called bias or intercept, and $\phi(\mathbf{x}_i)$ is known as the basis function matrix:

$$\phi^T(\mathbf{x}_n) = \left(1, x_{1,n}, x_{1,n}^2, \ldots, x_{1,n}^M, \ldots, x_{H,n}^M\right) \tag{14}$$

We apply a fully Bayesian approach, similarly as we did with the noise modeling in Section 2.4. Instead of considering the coefficients $\mathbf{w}_i$ as constant, we assume they are probabilistic variables, drawn from a normal distribution. Putting prior distributions on them, we can use the Bayes rule to calculate posterior distributions after knowing pair of inputs $\mathbf{x}_n$ and target values $t_{i,n}$. With these posteriors and given a new scenario $\tilde{\Upsilon}_{new}$, we can calculate the marginal distribution of the predicted target value using the Bayes rule [35]. For this model, we make the following assumptions:

- Gaussian noise: We will consider a Gaussian noise:

$$\epsilon_i \sim \mathcal{N}\left(0, \sigma_{\epsilon_i}^2\right) \tag{15}$$

and as priors of that noise, we take for each parameter i:

$$\sigma_{\epsilon_i}^2 \sim IG(A, B) \tag{16}$$

where A is the shape of the inverse gamma distribution (IG) and B its scale. It has been chosen $A = 5$ and $B = 10$, so that the distribution has a prior mean equal to 0.025 and a standard deviation equal to 0.0144.

- Input variables: the inputs are the adimensional scenario parameters, $\tilde{\Upsilon}$. Additionally, we will include two more variables that we believe that may represent the outputs:

$$\Upsilon'\{6\} = \frac{A_s}{N_a 2 R_f v_n} \tag{17a}$$

$$\Upsilon'\{7\} = \frac{A_s}{N_a 2 R_f} \tag{17b}$$

where $\Upsilon' = \{\Upsilon, \Upsilon'\{6\}, \Upsilon'\{7\},\}$ is the extended dimensional scenario parameters. Note that $\Upsilon'\{6\}$ is the ideal time to observe the complete area, whereas $\Upsilon'\{7\}$ is the total distance that each agent should travel so that the group observes all the area. Therefore, the inputs of the models are $\mathbf{x} \equiv \tilde{\Upsilon}'$.

- Polynomial order: third order polynomial, that is $M = 3$.

- Gaussian distribution: for each of the $\mathbf{w}_i$ parameter vector we will assume a posterior Gaussian distribution:

$$\mathbf{w}_i \sim \mathcal{N}(\mathbf{m}_i, \mathbf{S}_i) \tag{18}$$

with a prior distribution over both variables $\mathbf{m}_i$ and $\mathbf{S}_i$ also Gaussian. We assume a prior mean $\mathbf{m}_{i_0}^T = (0.5, 0, \ldots, 0)$ and a covariance matrix $\mathbf{S}_{i_0} = \text{diag}(0.25^2, 0.5^2, \ldots, 0.5^2)$. Note that the first elements correspond to the bias parameters, $w_{0,i}$.

3.2. Gaussian Process

The Gaussian Processes (GPs) are modeling methods that do not specify explicitly the functions that map the relationships between input variables and predicted values, as the BLR through the basis functions does. Instead, they set the covariances of the target values of the training set, as a function of their inputs [36]. This way, we define a distribution over a set of functions $y(x)$, whose joint distribution, when evaluated in the training inputs, is Gaussian. The GP is specified by selecting the covariance function, or kernel, between any two points of the data set. GPs are potentially more powerful than the BLR because the model is not restricted to very specific basis functions by setting its shape and parameters, but only general properties of the functions to be considered are fixed. On the other hand, GPs may become computationally infeasible if the data set is very large. It is important to remark that the main assumption for a GP is that the noise associated with the latent function f_i is Gaussian:

$$y_i = f_i(\mathbf{x}) + \epsilon_i \tag{19a}$$

$$\epsilon_i \sim \mathcal{N}\left(0, \sigma_{\epsilon_i}^2\right) \tag{19b}$$

If this condition is not met, the regression may make very incorrect predictions. For each of the variables of the control, we propose a GP, whose kernel will be selected among 10 possibilities.[1] For each possible kernel, its hyperparameters are optimized using a Bayesian optimization, see Section 4.2, in order to minimize the cross-validation loss error, defined as the mean squared error between the training data and the predicted values by the GP. As input variables, the same variables $\tilde{\Upsilon}'$ considered for the BLR will be used.

4. Optimization methods

4.1. Genetic algorithm optimization

As already mentioned, one of the most widely used methods to optimize complex algorithms in swarm robotics are the Genetic Algorithms. This is our first proposal to optimize the search algorithm.

4.1.1. Configuration of the GA

The main characteristics of the implemented GA are the following:

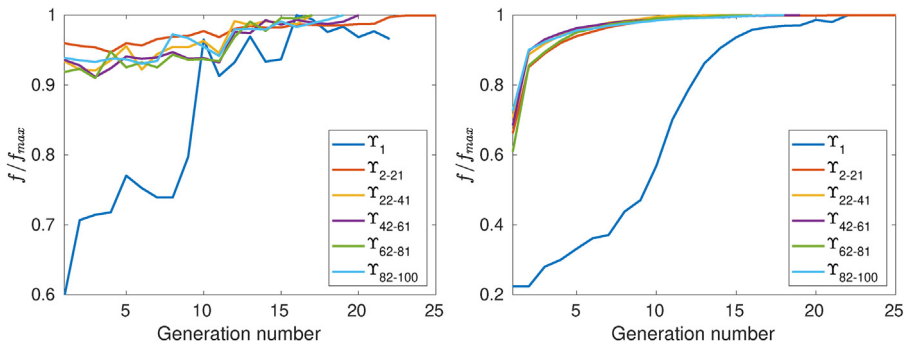
- *Chain of genes:* vector made up by the 23 parameters of the algorithm. Each of the genes is normalized with its limits, and therefore its range is [0,1].
- *Population:* for each generation, the population is made up by 50 members.
- *Initial population:* half of the initial population is taken from the two closest scenarios already optimized, 12 from each one. To select those members, a trade-off between fitness and genetic diversity is considered. The rest is taken as a prediction of the model, Eq. (12). Since the model is probabilistic, these 26 members are drawn from the normal distribution of the model.
- *Fitness function:* to evaluate each member, Eq. (11) is used.
- *Reevaluation:* every generation, the best member is retested if it has been tested less than 100 trials.
- *Crossover and mutation:* the members will be combined using the roulette-wheel technique, with probability proportional to the fitness function. Same pairs of members in each generation are forbidden. Being two members selected for crossover, a member will be created by combining each gene g_i of them:

$$g_i' = r_i \cdot g_i^1 + (1 - r_i) \cdot g_i^2 \tag{20}$$

where r_i is a random real number between 0 and 1, different for each gene g_i. The mutation is applied with a probability of $(1 - \bar{d}) \cdot P_{\max}$, where $\bar{d}$ is the diversity of the population divided by the diversity of a random population, and $P_{max} = 0.1$ is the maximum probability. The mutation then is applied as per:

$$g_i'' = g_i' + (0.5 - r_{i_{mut}}) \tag{21}$$

[1] Squared Exponential, Exponential, Matern 3/2, Matern 5/2, Rational Quadratic Kernel, ARD Squared Exponential, ARD Exponential, ARD Matern 3/2, ARD Matern 5/2 and ARD Rational Quadratic.



(a) Best member (b) Mean of the population

Fig. 4. Fitness of the best member and the mean of the population as a function of the generation number, divided by the maximum finally achieved, and averaged along ranges of scenarios, during the genetic algorithm optimization. Recall that the initial population is specialized and therefore, a good best member from the beginning is very likely.

where again $r_{i_{mut}}$ is a random real number between 0 and 1. The offspring is then made up by 50 new individuals.

- *Next generation selection:* after evaluating the new individuals, elitism is used to select the 50 best members for the next generation from the 100 available members.
- *Stopping criteria:* the optimization is stopped when one of these criteria is met:
  - Maximum number of generations (30) has been reached.
  - Maximum time for each optimization (24 hours) has been reached.
  - Maximum number of generations (10) without an improvement higher than a 10% of the best member has been reached.
  - Maximum number of generations (5) without an improvement higher than a 10% of the mean fitness of the population has been reached.

4.1.2. Selection of the next scenario

Given the BLR to model each parameter, we want to choose the scenarios to be optimized in a principled way. In these cases, the most sensible procedure is to choose the scenario for which there is a higher uncertainty in the model. However, there are two reasons that suggest not to proceed in this way:

- The optimization starts with an initial population, made up by members of near scenarios[2] and the predictions of the model. If a new scenario is further from the already optimized ones, it is more likely that the initial population is not appropriate. A bad initial population would likely lead to longer optimization processes.
- Scenarios with higher search areas, higher number of agents, or lower sensor footprints or velocities, will need more computational times.

Because of this, given a data set with j elements, we propose an evaluating criterion to choose the next scenario $j+1$ among a set of candidates by taking into account the uncertainty of the model, the estimated simulation time, and the distance to the already optimized scenarios. Selecting them taking into account a trade-off between them, leads to faster optimizations, combining exploitation of close scenarios, with exploration of further ones.

4.1.3. Complete process of optimization

The complete process of optimization follows the steps defined in Algorithm 1. We decided to limit the number of scenarios to be optimized to $n\_s\ cenarios = 100$. Note that for the first scenario, we established wider ranges for the parameters because we did not know them previously. Once the first scenario was optimized, the ranges can be adjusted so that the mean of the population lies on the mean of the ranges.

Algorithm 1. Genetic algorithm scenarios optimization

```
range_algorithm ⟵ broad
n_population ⟵ 50
Υ̃_1 ⟵ {0.5, 0.5, 0.5, 0.5, 0.5}
Ω̃*_1 ⟵ Optimize(Υ̃_1)
Model M
M ⟵ priors
M ⟵ UpdateModel(M, Υ̃_0, Ω̃*_0)
range_algorithm ⟵ narrow
n_scenarios ⟵ 99
for i ⟵ 2, n_scenarios + 1 do
    Υ̃_i ⟵ GetNewScenario(Υ̃_{j=1,...,i−1}, M)
    Ω̃*_i ⟵ Optimize(Υ̃_i)
    M ⟵ UpdateModel(M, Υ̃_i, Ω̃*_i)
end for
```

4.1.4. Results of the optimization with the GA

We ran the complete process in a single personal computer[3] and it took 21 days and 1 hour of computational time for the 99 scenarios (the first scenario is considered separately), that is 5 hours and 6 minutes on average for each scenario. In Fig. 4(a) the fitness of the best member, divided by the maximum reached through all generations, as a function of the number of generations, has been represented. The first scenario and 5 different scenario intervals have been considered to average the data.

The first thing that calls our attention is the completely different behavior between the first scenario Υ_1 and the others. The first starts with the 60% of the maximum fitness and increases with the generations, whereas in the others the best member already achieves between 92% and 96% of the maximum fitness in the first generation. This is because for the scenarios Υ_{2-100} half of the initial population is taken from the closest scenarios already optimized, and the other half from a prediction from the model; due to this, the initial population is already specialized and therefore, it is very likely that at least one of the initial members presents a high performance. This situation is also reinforced since every time we choose the next scenario to optimize, we favor scenarios close to

[2] The distance between scenarios is measured as the euclidean distance between two tuples of scenario parameters $\tilde{\Upsilon}_1$ and $\tilde{\Upsilon}_2$.

[3] Intel i7-6700k 4200Mhz, 16Gb DDR4 3200 Mz. The 4 available cores were used. The simulator has been coded in Matlab, using the available machine learning libraries.

scenarios already optimized, being the initial population very likely to be well suited for that new one.

Note that from one generation to the next, a drop in the performance may take place due to the reevaluation of the best member.

The optimization for the first scenario took 4.75 times longer than the average for the rest of the scenarios. Thanks to selecting specialized populations and choosing the sequence of scenarios in a convenient way, the overall optimization time was notably reduced. The reduction time is due to the combination of two effects:

- From the first generation, the population for scenarios Υ_{2-100} presents a higher performance, being the simulation times shorter as a consequence (the agents finish the search in less time).
- The optimization converges earlier. The mean number of generations for the scenarios Υ_{2-100} is 13.0, whereas for Υ_1 the optimization needed 22 generations.

It could be argued that based on Fig. 4(a) the optimization process fails, since there is barely an improvement of the best member. The optimization could therefore be skipped and the best member of the population in the first generation could be chosen. In Fig. 4(b), the mean fitness of the population has been represented as a function of the generation number, also grouped for different ranges of scenarios. It is clear that during the optimization, there is in fact a valuable optimization process, with an increment of the mean fitness between 37% and 61%. This graph justifies the utility of the optimization: although there is not much difference between the best solution before and after the optimization, there is indeed a relevant improvement in the mean of the population. Since part of this population will be taken for another initial population in a different scenario, it will help to speed up the GA.

4.2. Bayesian optimization

The Bayesian optimization (BOpt) method [37] is a principled method of finding suboptimal solutions taking into account the uncertainty of the model and the best expected improvement of an evaluation function. It models that function (in this case, the fitness, Eq. (11)) using a GP, and evaluates sequentially configurations picking them so that the so-called acquisition function is minimized, which considers a trade-off between exploration of new solutions, and exploitation of already-known good configurations. The BOpt is suitable for problems in which the evaluation has a high cost (either computational or experimental) and the objective function is non-convex. However, it has the drawback that it does not deal well with high dimensional problems. Another element to consider is that the search space must be bounded, which is not relevant in our case since we have already established a range of values. Therefore, the inputs for the BOpt are limited between 0 and 1.

For the BOpt, instead of using the BLR to model the suboptimal values among the scenarios, we use GPs. Each scenario is sequentially selected as done for the GA. Initially, the first 10 configurations to be evaluated are drawn from predictions of the GP. The optimization runs until there is no improvement higher than 10% in the last 200 tested configurations, or the maximum number of configurations, equal to 1600, are reached. The complete process of optimization follows a similar procedure presented in Algorithm 1, but without a first scenario optimized independently, as we did for the GA (no wide ranges are needed, since we already known them after running the GA optimization). The model M is a GP, instead of a BLR, and the step 12, *Optimize*($\tilde{\Upsilon}_i$), is a BOpt, instead of a GA.

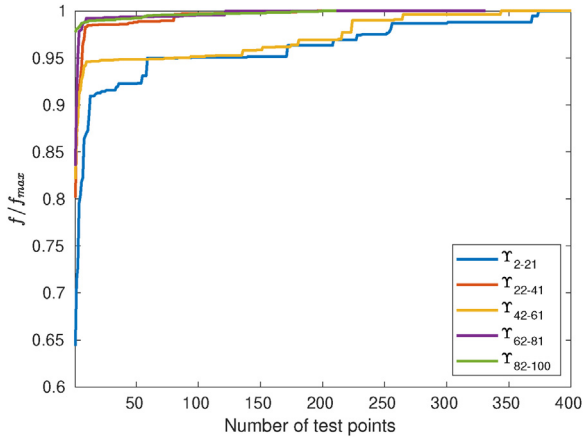


Fig. 5. Ideal fitness of the best solution found in the BOpt, as a function of the number of tested points, divided by the maximum fitness, and averaged along ranges of scenarios.

4.3. Results of the BOpt

The BOpt was run in the same personal computer, and it took 39 days and 23 hours to optimize the 100 scenarios, which is almost two times longer. For each scenario, 9 hours and 36 minutes were needed on average. In Fig. 5 the fitness function of the best found solution has been represented as a function of the number of the tested configurations, and divided by the best found solution in the complete process of optimization. Again, the values have been grouped in 5 ranges of scenario numbers. For the BOpt, there is a higher improvement during the optimization, typically starting between the 80% and the 85% of the maximum fitness, and reaching it after between 200 and 500 evaluations. The BOpt requires more time mainly because the convergence is slower, given the selected criteria. Also, every time the BOpt gets a new point, it updates the internal GP to select the next configuration. Since it depends on 23 predictors (i.e. the 23 variables of the algorithm), it takes time and the process is slowed down.

5. Comparison of the methods

Once we obtain two suboptimal configuration data sets, we first compare both regression methods, the BLR and the GPs, to check which method fits better. Secondly, we select the best regression method for both data sets and check both optimization models in new scenarios, comparing the efficiencies reached with random configurations. Finally, we model the performance in order to predict it for any new scenario.

5.1. Regression comparison

We want to compare how well both regression methods, the BLR and the GP, adjust to the training data, obtained by the GA and the BOpt. To do that, we compare the loss function of both regressions, calculated as the mean squared error between the training data (suboptimal values of each scenario) and the values predicted by the regression:

$$\mathcal{L}\left(\mathbf{y}_i, \hat{\mathbf{y}}_i\right) = \frac{1}{N}\sum_{n=1}^{N}(y_{i,n} - \hat{y}_{i,n})^2 \tag{22}$$

where i indicates the parameter number, $i = 1, \ldots, 23$, $n = 1, \ldots, 100$ is the scenario number, $\mathbf{y}_i$ is the data set vector, and $\hat{\mathbf{y}}_i$ is the vector with the values predicted by the regression. In Table 4 the mean values of the loss function ($\mu_{\mathcal{L}}$) of the 23 parameters has been presented, with its standard deviation ($\sigma_{\mathcal{L}}$) for the GA and the BOpt,

Table 4
Loss function (mean $\mu_{\mathcal{L}}$ and deviation $\sigma_{\mathcal{L}}$) of both regression methods, the Bayesian Linear Regression (BLR) and the Gaussian Process (GP), for the optimization using the Genetic Algorithm (GA) and the Bayesian Optimization (BOpt).

	GA		BOpt	
	$\mu_{\mathcal{L}}$	$\sigma_{\mathcal{L}}$	$\mu_{\mathcal{L}}$	$\sigma_{\mathcal{L}}$
BLR	$2.7{\cdot}10^{-2}$	$0.83{\cdot}10^{-4}$	$6.1{\cdot}10^{-2}$	$2.36{\cdot}10^{-4}$
GP	$0.5{\cdot}10^{-2}$	$0.88{\cdot}10^{-4}$	$0.03{\cdot}10^{-2}$	$0.02{\cdot}10^{-4}$

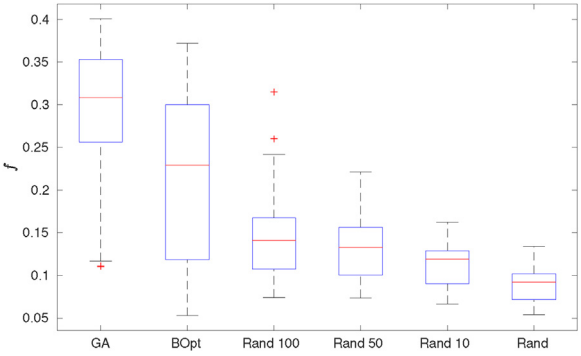


Fig. 6. Fitness distribution for 120 new random scenarios, configuring the algorithm with the Genetic Algorithm (GA), the Bayesian Optimization (BOpt), taking the best of 100 random solutions (Rand 100), the best of 50 (Rand 50), the best of 10 (Rand 10), and a random solution (Rand). For each scenario, the fitness of the configuration has been analyzed as per Sections 2.4 and 2.5.

using the BLRs and the GPs. It is clear that the GPs are more capable to adapt to the data. In the case of the GA, the GP approaches the data set with a mean loss function 5 times lower than the BLR, and in the case of the BOpt, the mean is reduced 20 times. On the other hand, if we study the coefficients $\mathbf{w}_i$ obtained by the BLR we can easily see the influence of the scenario on the suboptimal configurations. For instance, having a look on the coefficients of α_D (see original paper) related with the search area size per agent, A_s/N_a, we can learn that the higher this size is, the higher the coefficient α_D gets, implying that the agents try more to spread out over the search area.

5.2. Comparison of optimization processes

As we have seen in the last section, the regression based on GPs adjusts better to the training data than the BLR, although this last approach is useful to understand how the algorithm works. We will then use the GP regression to approach the suboptimal solutions found during both the GA and the BOpt processes.

We want to compare the performance of the algorithm configuring it using the data set obtained with the GA and the BOpt. We generate 120 new scenarios and test the algorithm with both configurations, and compare the results with the performances obtained using random configurations. In Fig. 6 the fitness function distribution for the new 120 scenarios has been represented. Each new scenario has been analyzed as explained in Sections 2.4 and 2.5. The first distribution corresponds to the GA configuration, whereas the second one corresponds to the BOpt. The third one, labeled as *Rand 100*, is the best fitness obtained from 100 random configurations, generated and tested independently for each scenario. *Rand 50*, *Rand 10* and *Rand* correspond to the fitness function obtained from 50, 10 and 1 random configurations, again different for each scenario.

As it can be seen, the GA reaches the highest mean fitness value, equal to 0.31, with the 50% of the values between 0.26 and 0.35. With the BOpt, a mean fitness value of 0.23 is obtained, lower than that of the GA. Also, the variance of the performances is larger, getting 1/2 of the values between 0.23 and 0.30. This result shows that the GA is a better approach than the BOpt for this algorithm, since it achieves higher fitness values and needs less computational time. The second best approach is the best solution found from 100 random configurations, generated independently for each scenario. A mean fitness value of 0.14 is reached, with a range of values between 0.11 and 0.17. Then, the 100 Random configuration only achieves on average the 45% of the fitness achieved by the GA, and the 61% by the BOpt. Moreover, every time we have a new scenario we have to simulate 100 random configurations, which takes on average 3 hours and 54 minutes. If we use one random configuration we obtain very poor results, with an average fitness of 0.09. There is an improvement if we select the best of 10 random configurations, achieving a mean fitness of 0.12, for which we would need 23 minutes on average to compute it. Increasing the number of random configurations to 50 or 100, does not improve substantially the fitness. Considering this progression, we can assume that the ceiling of the random search will be close to the Rand 100 distribution, with much lower performances compared to the GA and the BOpt. Moreover, for each new scenario we propose, we obtain the configuration using the regression within few seconds, a very short time compared with a random search, and with better performance results.

Table 5
Probability that the GA and the BOpt (on the left column) perform better than other method (upper row).

	BOpt	Rand 100	Rand 50	Rand 10	Rand
GA	0.74	0.93	0.94	0.95	0.99
BOpt	–	0.71	0.73	0.78	0.89

Instead of comparing the fitness distribution for the test scenarios, we can compare, scenario by scenario, the efficiency distribution of each of the methods proposed in Fig. 6. Having assumed that the efficiency is drawn from a normal distribution, and having identified a mean and a standard deviation as explained in Section 2.4, the probability that one of the method is better than other can be calculated with Eqs. (9) and (10). In Table 5, the probability that one configuration performs better than other has been presented. Note that the configuration modeled by the GA is very likely that it performs better than any other.

5.3. Efficiency and fitness regression

As we see in Fig. 6 and in Table 5, the regression of suboptimal parameters using GPs, tuned with the results of the GA, achieves the highest performances. We are also interested in modeling the efficiency and the fitness functions so that given any scenario Υ we can predict them. This implies that we would be also able to predict the time in which the search task will be carried out, and its variance as result of the unknown initial conditions. Recall that to calculate the fitness, it has been considered through the coefficient β_e in Eq. (11) a reduction of the 75% of the efficiency in case the search is not completed before the agents run out of energy. This breaks the smoothness of the fitness as a function of the scenario parameters, being then more difficult to model. To overcome this issue, we model separately the efficiency, its standard deviation due to the initial conditions, and the probability of finishing the search, P_s. This last variable is estimated dividing the number of trials in which the search has been completed by the total number of trials. To train the three GPs, we use the results of 120 simulations presented in Fig. 6. The loss function of these 3 GPs shows a good adjustment to the data set for the efficiency and the standard deviation. However, P_s is more difficult to model, presenting a loss function equal to 0.014.

Table 6
Three scenario examples with measured and predicted values for the efficiency (E), its standard deviation (σ_E), the probability of finishing the search (P_s), the fitness (f) and the interval of time in which the search is going to last, with a 95% of confidence ($time_{95}$).

	Scenario 1		Scenario 2		Scenario 3	
A_s/N_a [m^2]	7020		9750		14,354	
N_a	17		4		8	
v_n [m/s]	9.7		8.6		7.0	
R_f [m]	16.0		6.4		6.3	
f_A	0.95		0.53		0.32	
	Measured	Predicted	Measured	Predicted	Measured	Predicted
E	0.39	0.38	0.50	0.50	0.64	0.60
σ	0.034	0.037	0.024	0.033	0.021	0.020
P_s	1.00	1.00	0.17	0.50	0.00	0.00
f	0.34	0.33	0.18	0.25	0.16	0.13
$time_{95}$ [s]	52–54	57–61	173–179	173–184	218–222	263–274

Once we have the three models, we are able to predict the values for 100 new scenarios and compare the predicted values with the measured ones. With the predictions, we calculate the estimated fitness with Eq. (11). As an example, in Table 6 three random scenarios have been presented, with a comparison between measured and predicted values for the efficiency, its standard deviation, the probability of finishing the task and the fitness. Using Eqs. (2) and (1) we can calculate the interval of time with a 95% of confidence (under the assumption of a normal distribution of the efficiency) in which the search is going to last. For scenarios in which the search is either always or never completed, the predictions are more accurate (Scenarios 1 and 3), since the probability of completing the mission is well modeled. For the Scenario 2, in which in some cases the task is not completed, the model predicts worse P_s and therefore the fitness.

6. Conclusions and future works

Many of robotic swarms are controlled by behaviors, whose internal variables and transition parameters must be tuned. The more parameters there are, the more difficult is to find appropriate values. Moreover, there may exist external parameters not selectable by the designer, such as the work area or the maximum number of robots, on which the configuration of the swarm depends. This means that the tuning of the control depends on the scenario parameters.

In this work we have compared two different methods to optimize a complex algorithm with a high dimensional configuration space (23 variables), showing in both cases feasibility and good results. Since the suboptimal configurations depend on external parameters that cannot be selected (here the so-called scenario parameters), several optimizations must be performed to be able to properly tune it for a broad range of cases. Once we have obtained a data set with suboptimal configurations for 100 scenarios, we have also studied two regression methods to approximate them for any other scenario. This way we can configure the algorithm on-line, achieving good performances for the task considered. We have compared this approach with a random search of 100, 50, 10 and 1 configurations, showing that these lasts reach much lower performances and must be calculated off-line.

Our solution not only may be implemented for global tasks, but also for subtaks belonging to a more global mission, or even for single behaviors. This way, when an agent must execute a specific task, and the optimal configuration of the control depends on the status of the global mission and on the state of the other agents, it can immediately be configured.

Our approach comes to a prize: it requires massive and very fast simulations. Although efforts have been made to reduce the computational load, the optimization process requires long periods of time. In order to speed the simulations up, another numerical scheme for the pheromones can be implemented, for instance a Crank-Nicolson scheme, which allows higher time steps, reducing therefore the computational times. Also, moving from an interpreted programming language (Matlab) to a compiled one (such as C++) could reduce the simulation times.

Also for this reason, the simulated world and the model of the agent must be simplified and if the algorithm was tested in real robots, it would probably suffer from the reality gap. The probabilistic approach helps to avoid overfitting, reducing the problem. Another possible approach to reduce the reality gap is to include more accurate simulations after a first approximation with very simplified models. Once we have a regression with rough optimal values, we can use this prior knowledge to fine tune them afterwards using more accurate simulations, but reducing the number of trials. Also, we can repeat again this procedure but with tests on real robots to finally eliminate the reality gap. In this case, probably the most convenient optimization method would be the BOpt, since allows to reduce the number of tests, which is critical when field experiments are involved. Intermediate steps can be also added in case the mission is very complex and the number of parameters involved is very high; one can start with very simple models, and step by step, finely tune the algorithm with increasing levels of accuracy until reaching the experimental level. Dividing a global task into subtasks, and optimizing them individually may also be a convenient approach.

Acknowledgements

We would like to thank SAVIER (Situational Awareness Virtual EnviRonment) Project, which is both supported and funded by Airbus Defence & Space. The research leading to these results has received funding from the RoboCity2030-III-CM project (Robótica aplicada a la mejora de la calidad de vida de los ciudadanos. Fase III; S2013/MIT-2748), funded by Programas de Actividades I+D en la Comunidad de Madrid and cofunded by Structural Funds of the EU, and from the DPI2014-56985-R project (Protección Robotizada de Infraestructuras Críticas, PRIC) funded by the Ministerio de Economía y Competitividad of Gobierno de España.

References

[1] E. Bonabeau, M. Dorigo, G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems, 1, Oxford University Press, 1999.
[2] J. Deneubourg, S. Goss, G. Sandini, F. Ferrari, P. Dario, Self-organizing collection and transport of objects in unpredictable environments, Japan-USA Symposium on Flexible Automation (1990) 1093–1098.
[3] C.R. Kube, H. Zhang, Collective robotics: from social insects to robots, Adapt. Behav. 2 (2) (1993) 189–218.
[4] M.J. Mataric, Designing emergent behaviors: from local interactions to collective intelligence, Proceedings of the Second International Conference on Simulation of Adaptive Behavior (1993) 432–441.
[5] E. Şahin, Swarm robotics: from sources of inspiration to domains of application, in: International Workshop on Swarm Robotics, Springer, 2004, pp. 10–20.

[6] M. Brambilla, E. Ferrante, M. Birattari, M. Dorigo, Swarm robotics: a review from the swarm engineering perspective, Swarm Intell. 7 (1) (2013) 1–41.
[7] A. Martinoli, K. Easton, W. Agassounon, Modeling swarm robotic systems: a case study in collaborative distributed manipulation, Int. J. Robot. Res. 23 (4–5) (2004) 415–436.
[8] A.F. Winfield, W. Liu, J. Nembrini, A. Martinoli, Modelling a wireless connected swarm of mobile robots, Swarm Intell. 2 (2) (2008) 241–266.
[9] H. Hamann, H. Wörn, A framework of space-time continuous models for algorithm design in swarm robotics, Swarm Intell. 2 (2) (2008) 209–239.
[10] K. Lerman, A. Galstyan, A. Martinoli, A. Ijspeert, A macroscopic analytical model of collaboration in distributed robotic systems, Artif. Life 7 (4) (2001) 375–393.
[11] W. Liu, A.F. Winfield, Modeling and optimization of adaptive foraging in swarm robotic systems, Int. J. Robot. Res. 29 (14) (2010) 1743–1760.
[12] E. Azketa, J.J. Gutierrez, M. Di Natale, L. Almeida, M. Marcos, Permutational genetic algorithm for the deployment and scheduling of distributed real time systems, Rev. Iberoam. Autom. Inform. Ind. 10 (3) (2013) 344–355.
[13] V. Trianni, S. Nolfi, M. Dorigo, Evolution, self-organization and swarm robotics, in: Swarm Intelligence, Springer, 2008, pp. 163–191.
[14] P. Gaudiano, E. Bonabeau, B. Shargel, Evolving behaviors for a swarm of unmanned air vehicles, in: Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005, SIS 2005, IEEE, 2005, pp. 317–324.
[15] A.L. Nelson, E. Grant, J.M. Galeotti, S. Rhody, Maze exploration behaviors using an integrated evolutionary robotics environment, Robot. Auton. Syst. 46 (3) (2004) 159–173.
[16] G. Francesca, M. Brambilla, A. Brutschy, V. Trianni, M. Birattari, Automode: a novel approach to the automatic design of control software for robot swarms, Swarm Intell. 8 (2) (2014) 89–112.
[17] J.P. Hecker, M.E. Moses, Beyond pheromones: evolving error-tolerant, flexible, and scalable ant-inspired robot swarms, Swarm Intell. 9 (1) (2015) 43–70.
[18] R. Brooks, A robust layered control system for a mobile robot, IEEE J. Robot. Autom. 2 (1) (1986) 14–23.
[19] R.C. Arkin, Behavior-Based Robotics, MIT Press, 1998.
[20] T. Balch, R.C. Arkin, Behavior-based formation control for multirobot teams, IEEE Trans. Robot. Autom. 14 (6) (1998) 926–939.
[21] G. Antonelli, F. Arrichiello, S. Chiaverini, Flocking for multi-robot systems via the null-space-based behavioral control, Swarm Intell. 4 (1) (2010) 37.
[22] M. Proetzsch, T. Luksch, K. Berns, Development of complex robotic systems using the behavior-based control architecture ib2c, Robot. Auton. Syst. 58 (1) (2010) 46–67.
[23] G. Antonelli, F. Arrichiello, S. Chiaverini, The nsb control: a behavior-based approach for multi-robot systems, Paladyn, J. Behav. Robot. 1 (1) (2010) 48–56.
[24] P. Garcia-Aunon, A. Barrientos Cruz, Comparison of heuristic algorithms in discrete search and surveillance tasks using aerial swarms, Appl. Sci. 8 (5) (2018) 1–31.
[25] J.M. Sullivan, Revolution or evolution? The rise of the UAVS, in: 2005 International Symposium on Technology and Society, 2005. Weapons and Wires: Prevention and Safety in a Time of Fear. ISTAS 2005. Proceedings, IEEE, 2005, pp. 94–101.
[26] J.A. Sauter, R. Matthews, H. Van Dyke Parunak, S.A. Brueckner, Performance of digital pheromones for swarming vehicle control, in: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, ACM, 2005, pp. 903–910.
[27] R.R. McCune, G.R. Madey, Control of artificial swarms with DDDAS, Proc. Comput. Sci. 29 (2014) 1171–1181.
[28] D.K. Sutantyo, S. Kernbach, P. Levi, V.A. Nepomnyashchikh, Multi-robot searching algorithm using lévy flight and artificial potential field, in: 2010 IEEE International Workshop on Safety Security and Rescue Robotics (SSRR), IEEE, 2010, pp. 1–6.
[29] W. Liu, Y.E. Taima, M.B. Short, A.L. Bertozzi, Multi-scale collaborative searching through swarming, in: ICINCO (2), 2010, pp. 222–231.
[30] S. Waharte, A.C. Symington, N. Trigoni, Probabilistic search with agile uavs, in: ICRA, 2010, pp. 2840–2845.
[31] Y. Altshuler, V. Yanovsky, I.A. Wagner, A.M. Bruckstein, Efficient cooperative search of smart targets using uav swarms1, Robotica 26 (4) (2008) 551–557.
[32] T. Stirling, S. Wischmann, D. Floreano, Energy-efficient indoor search by swarms of simulated flying robots without global information, Swarm Intell. 4 (2) (2010) 117–143.
[33] A. Jevtić, A. Gutiérrez, Distributed bees algorithm parameters optimization for a cost efficient target allocation in swarms of robots, Sensors 11 (11) (2011) 10880–10893.
[34] K.P. Murphy, Conjugate Bayesian Analysis of the Gaussian Distribution, Def 1 ($2\sigma 2$), 2007, pp. 16.
[35] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.
[36] C.E. Rasmussen, C.K. Williams, Gaussian Processes for Machine Learning, vol. 1, MIT Press Cambridge, 2006.
[37] E. Brochu, V.M. Cora, N. De Freitas, 1 A Tutorial on Bayesian Optimization of Expensive Cost Functions, With Application to Active User Modeling and Hierarchical Reinforcement Learning, 2599, 2010, pp. 1–49, arXiv preprint arXiv:1012.

PhD Candidate Pablo García Auñón develops his investigations since 2015 at the Centre for Automation and Robotics in Madrid, Spain. Previously, after working 3 years in the aeronautical industry in Germany and Spain, he got a Master's Degree in System Engineering and Control at the UNED. He studied earlier Aeronautical Engineering, aircraft specialization, at the UPM. His research lines are distributed robotics, and more specifically, the development of algorithms for aerial and ground robotic swarm in search, surveillance and exploration tasks.

Dr. Antonio Barrientos received the MSc in Automation and Electronics from the Technical University of Madrid (UPM) in 1982, and the PhD in Robotics from the same University in 1986. In 2002 he obtained the MSc Degree in Biomedical Engineering by National Distance Education University (UNED). Since 1988 he is Professor on robotics, computers and control engineering at the Technical University of Madrid. He has worked in robotics for more than 30 years, developing industrial and service robots for different areas. Antonio Barrientos is currently the head of the Robotics and Cybernetics Research Group (RobCib) of the Centre for Automation and Robotics (UPM-CSIC).