Sniffy Bug: A Fully Autonomous Swarm of Gas-Seeking Nano Quadcopters in Cluttered Environments

Bardienus P. Duisterhof[1] Shushuai Li[1] Javier Burgués[2] Vijay Janapa Reddi[3] Guido C.H.E. de Croon[1]

***Abstract*—Nano quadcopters are ideal for gas source localization (GSL) as they are safe, agile and inexpensive. However, their extremely restricted sensors and computational resources make GSL a daunting challenge. We propose a novel bug algorithm named 'Sniffy Bug', which allows a fully autonomous swarm of gas-seeking nano quadcopters to localize a gas source in unknown, cluttered, and GPS-denied environments. The computationally efficient, mapless algorithm foresees in the avoidance of obstacles and other swarm members, while pursuing desired waypoints. The waypoints are first set for exploration, and, when a single swarm member has sensed the gas, by a particle swarm optimization-based (PSO) procedure. We evolve all the parameters of the bug (and PSO) algorithm using our novel simulation pipeline, 'AutoGDM'. It builds on and expands open source tools in order to enable fully automated end-to-end environment generation and gas dispersion modeling, allowing for learning in simulation. Flight tests show that Sniffy Bug with evolved parameters outperforms manually selected parameters in cluttered, real-world environments. Videos: https://bit.ly/37MmtdL**

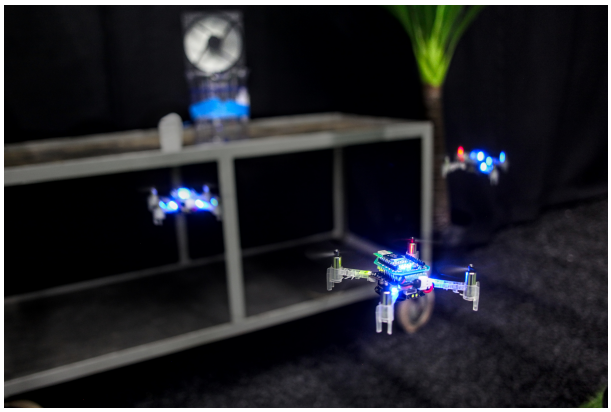


Fig. 1. A fully autonomous and collaborative swarm of gas-seeking nano quadcopters, finding and locating an isopropyl alcohol source. The source is visible in the background: a spinning fan above a can of isopropyl alcohol.

I. Introduction

Gas source localization (GSL) by autonomous robots is important for search and rescue and inspection, as it is a very dangerous and time-consuming task for humans. A swarm of nano quadcopters is an ideal candidate for GSL in large, cluttered, indoor environments. The quadcopters' tiny size allows them to fly in narrow spaces, while operating as a swarm enables them to spread out and find the gas source much quicker than a single robot would.

To enable a fully autonomous gas-seeking swarm, the nano quadcopters need to navigate in unknown, cluttered, GPS-denied environments by avoiding obstacles and each other. Currently, indoor swarm navigation is still challenging and an active research topic even for larger quadcopters ($>$ 500 grams) [1], [2]. State-of-the-art methods use heavy sensors like LiDAR and high-resolution cameras to construct a detailed metric map of the environment, while also estimating the robot's position for navigation with Simultaneous Localization And Mapping (SLAM, e.g., ORB-SLAM2 [3]). These methods do not fit within the extreme resource restrictions of nano quadcopters. The payload of nano quadcopters is in the order of grams, ruling out heavy, power-hungry sensors like LiDAR. Furthermore, SLAM algorithms typically require GBs of memory [4] and need considerable processing power. One of the most efficient implementations of SLAM runs real-time (18.21 fps) on an ODROID-XU4 [5], which has a CPU with a 4-core @ 2GHz plus a 4-core @1.3GHz. These properties rule out the use of SLAM on nano quadcopters such as the Bitcraze Crazyflie, which has an STM32F405 processor with 1MB of flash memory and a single core @168MHz. As a result of the severe resource constraints, previous work has explored alternative navigation strategies. A promising solution was introduced in [6], in which a bug algorithm enabled a swarm of nano quadcopters to explore unknown, cluttered environments and come back to the starting location.

Besides navigating, the swarm also needs a robust strategy to locate the gas source, which by itself is a highly challenging task. This is mostly due to the complex spreading of gas in cluttered environments. Moreover, current sensors have poor quality compared to animals' smelling capabilities [7], which is further complicated by the propellers' own down-wash [8].

Various solutions to odor source localization have been studied. Probabilistic GSL strategies [9], [10] usually internally maintain a map with the probabilities for the odor source location and often simulate the gas distribution. This is computationally challenging for nano quadcopters, a situation that deteriorates when they have to operate in environments with complex shapes, obstacles and a complex airflow field. In contrast, bio-inspired finite-state machines [11], [12] have very low computational requirements, though until now they

[1] Delft University of Technology, [2] University of Barcelona, [3] Harvard University. Email: b.p.duisterhof@gmail.com or g.c.h.e.decroon@tudelft.nl

have focused on deploying a single agent [13]. Moreover, reinforcement and evolutionary learning approaches have been investigated, mostly in simulation [14], [15], [16], [17], [18], but a few works also transferred the learned policy to obstacle-free environments in the real world [19], [20]. In [21], a policy was trained for light seeking and transferred from simulation to a real environment with simple obstacles. A limiting factor for approaches that learn in simulation is that gas dispersion modeling has been a time-intensive process, requiring domain knowledge for accurate modeling. Only few environments have been made available to the public [22], [23], whereas learning algorithms require many different environments.

Due to the problem's difficulty in the real world, the experiments typically involve a single robot seeking for an odor source in an obstacle-free environment of modest size, e.g., in the order of 4×4 m [24], [25], [26], [27]. Few experiments have been performed in larger areas involving multiple robots. The use of particle swarm optimization [28] is promising, as it can deal with local maxima in gas concentration that arise in more complex environments. Closest to our work is an implementation of PSO on a group of large ($2.95\,\mathrm{kg}$), outdoor flying quadcopters [29] , using LiDAR and GPS for navigation.

In this article, we introduce a novel PSO-powered bug algorithm, *Sniffy Bug*, to tackle the problem of GSL in challenging, cluttered, and GPS-denied environments. The nano quadcopters execute PSO by estimating their relative positions and by communicating observed gas concentrations to each other using onboard Ultra-Wideband (UWB) [30]. In order to optimize the parameters of Sniffy Bug with an artificial evolution, we also develop and present the first fully Automated end-to-end environment generation and Gas Dispersion Modeling pipeline, which we term *AutoGDM*. We validate our approach with robot experiments in cluttered environments, showing that evolved parameters outperform manually tuned parameters. This leads to the following three contributions:

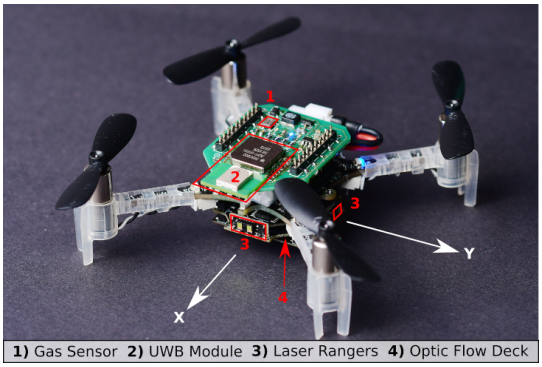
1) The first robotic demonstration of a swarm of autonomous nano quadcopters locating a gas source in unknown, GPS-denied environments with obstacles.
2) A novel, computationally highly efficient bug algorithm, Sniffy Bug, of which the parameters are evolved for PSO-based gas source localization in unknown, cluttered and GPS-denied environments.
3) The first fully automated environment generation and gas dispersion modeling pipeline, AutoGDM.

In the remainder of this article, we explain our methodology (Section II), present simulation and flight results (Section III), and draw conclusions (Section IV).

II. Method

A. System Design

Our $37.5\,\mathrm{g}$ Bitcraze Crazyflie [31] nano quadcopter (Figure 2), is equipped with sensors for waypoint tracking, obstacle avoidance, relative localization, communication and gas sensing. We use the optical flow deck and IMU sensors for estimating the drone's state and tracking waypoints. Additionally, we use the Bitcraze multiranger deck with four laser range sensors in the drone's positive and negative x and y axis (Figure 2), to sense and avoid obstacles. Finally, we have designed a custom, open-source PCB, capable of gas sensing and relative localization. It features a Figaro TGS8100 MEMS MOX gas sensor, which is lightweight, inexpensive, low-power, and was previously deployed onboard a nano quadcopter [32]. We use it to seek isopropyl alcohol, but it is sensitive to many other substances, such as carbon monoxide. The TGS8100 changes resistance based on exposure to gas, which can be computed according to Equation 1.



1) Gas Sensor **2)** UWB Module **3)** Laser Rangers **4)** Optic Flow Deck

Fig. 2. A $37.5\,\mathrm{g}$ nano quadcopter, capable of autonomous waypoint tracking, obstacle avoidance, relative localization, communication and gas sensing.

$$R_s = \left(\frac{V_c}{V_{RL}} - 1\right) \cdot R_L \tag{1}$$

Here R_s is the sensor resistance, V_c circuit voltage ($3.0\,\mathrm{V}$), V_{RL} the voltage drop over the load resistor in a voltage divider, and R_L is the load resistor's resistance ($68\,\mathrm{k\Omega}$). Since different sensors can have different offsets in the sensor reading, we have designed our algorithm not to need absolute measurements like a concentration in ppm. From now on, we report a corrected version of V_{RL}, where higher means more gas. V_{RL} is corrected by its initial low-passed reading, without any gas present, in order to correct sensor-specific bias.

For relative ranging, communication, and localization, we use a Decawave DWM1000 UWB module. Following [30], an extended Kalman filter (EKF) uses onboard sensing from all agents, measuring velocity, yaw rate, and height, which is fused with the UWB range measurements. It does not rely on external systems or magnetometers. Additionally, all agents are programmed to maintain constant yaw, as it further improves the stability and accuracy of the estimates.

B. Algorithm Design

The main idea behind Sniffy Bug is to attain high computational efficiency by combining PSO, which indicates search directions, with a bug algorithm, which navigates to these directions. We generate waypoints in the reference frame of each agent using PSO, based on the relative positions and gas readings of all agents. The reference frame of the agent is initialized just before takeoff, and is maintained using dead reckoning by fusing data from the IMU and optic flow sensors.

While the reference frame does drift over time, only the drift since the last seen 'best waypoint' in PSO will be relevant, as it will be saved in the drifted frame.

The waypoints are tracked using a bug algorithm that follows walls and other objects, moving safely around them. Each agent computes a new waypoint if it reaches within d_{wp} of its previous goal, if the last update was more than t_{wp} seconds ago, or when one of the agents smells a concentration superior to what has been seen by any agent during the run, and higher than the pre-defined detection threshold. A detection threshold is in place to avoid reacting based on sensor drift and noise. A timeout time, t_{wp}, is necessary, as the predicted waypoint may be unreachable (e.g., inside a closed room). We term each new waypoint, generated if one of the above criteria is met, a 'waypoint iteration' (e.g., waypoint iteration five is the fifth waypoint generated during that run).

1) Waypoint generation: Each agent computes its next waypoint according to Equation 2.

$$\mathbf{g}_{i,j} = \mathbf{x}_{i,j} + \mathbf{v}_{i,j} \tag{2}$$

Here $\mathbf{g}_{i,j}$ is the goal waypoint of agent i, in iteration j, $\mathbf{x}_{i,j}$ its position when generating the waypoint, and $\mathbf{v}_{i,j}$ its 'velocity vector'. The velocity vector is determined depending on the drone's mode, which can be either 'exploring', or 'seeking'. 'Exploring' is activated when none of the agents has smelled gas, while 'seeking' is activated when an agent has detected gas. During exploration, $\mathbf{v}_{i,j}$ is computed with Equation 3:

$$\mathbf{v}_{i,j} = \omega'(\mathbf{g}_{i,j-1} - \mathbf{x}_{i,j}) + r_r\left(\mathbf{r}_{i,j} - \mathbf{x}_{i,j}\right) \tag{3}$$

Here $\mathbf{g}_{i,j-1}$ is the goal computed in the previous iteration and $\mathbf{r}_{i,j}$ a random point within a square of size r_{rand} around the agent's current position. A new random point is generated each iteration. Finally, ω' and r_r are scalars that impact the behavior of the agent. $\mathbf{g}_{i,j}$ and $\mathbf{v}_{i,j}$ are initialized randomly in a square of size r_{rand} around the agent. Intuitively, Equation 3 shows the new velocity vector is a weighted sum of: 1) a vector toward the previously computed goal (also referred to as inertia), and 2) a vector towards a random point.

After smelling gas, i.e., one of the agents detects a concentration above a pre-defined threshold, we update the waypoints according to Equation 4.

$$\begin{aligned}\mathbf{v}_{i,j} = \omega(\mathbf{g}_{i,j-1} - \mathbf{x}_{i,j}) + \varphi_p\alpha_{i,j}\left(\mathbf{p}_{i,j} - \mathbf{x}_{i,j}\right) + \\ \varphi_g\beta_{i,j}\left(\mathbf{s}_j - \mathbf{x}_{i,j}\right)\end{aligned} \tag{4}$$

Here $\mathbf{p}_{i,j}$ is the waypoint at which agent i has seen its highest concentration so far, up to iteration j. $\mathbf{s}_j$ is the swarm's best seen waypoint, up to iteration j. $\alpha_{i,j}$ and $\beta_{i,j}$ are random values between 0 and 1, generated for each waypoint iteration for each agent. Finally, φ_p and φ_g are scalars that impact the behavior of the agent. So again, more intuitively, the vector towards the next waypoint is a weighted sum of the vectors towards its previously computed waypoint, the best seen position by the agent, and the best seen position by the swarm. As we will see later, this allows the swarm to converge to high concentrations of gas, whilst still exploring.

2) Waypoint tracking: Tracking waypoints in cluttered environments is hard due to the limited sensing and computational resources. Sniffy Bug is designed to operate at constant yaw, and consists of three states (Figure 3): 1) Line Following, 2) Wall Following, and 3) Attraction-Repulsion Swarming.

Line Following – When no obstacles are present, the agent follows a virtual line towards the goal waypoint, making sure to only move in directions where it can 'see' using a laser ranger, improving safety. The agent makes sure to stay within a distance d_{line} from the line, moving as shown in Figure 3.

Wall Following – When the agent detects an obstacle, and no other agents are close, it will follow the object similar to other bug algorithms [33]. Sniffy Bug's wall-following stage is visible in Figure 3. It is terminated if one of the following criteria is met: 1) a new waypoint has been generated, 2) the agent has avoided the obstacle, or 3) another agent is close. In case 1 and 2 line following is selected, whereas in case 3 attraction-repulsion swarming is activated. Figure 3 illustrates wall following in Sniffy Bug. The agent starts by computing $desiredlaser$, which is the laser direction that points most directly to the goal waypoint, laser 3 in this case. It then determines the initial search direction in the direction of the waypoint, anti-clockwise in this case. The agent now starts looking for laser rangers detecting a safe value (above d_{laser}), starting from lasers 3, in the anti-clockwise direction. As a result, we follow the wall in a chainsaw pattern, alternating between lasers 3 and 0. Next, the agent uses odometry to detect that it has avoided the obstacle, by exiting and re-entering the green zone, while getting closer to the goal waypoint.

Attraction-Repulsion Swarming – When the agent detects at least one other agent within a distance d_{swarm}, it uses attraction-repulsion swarming to avoid other agents and objects, while tracking its goal waypoint. This state is terminated when no agent is within d_{swarm}, selecting 'line following' instead. As can be seen in Figure 3 and Equations 5,6, the final commanded velocity vector is a sum of repulsive forces away from low laser readings and close agents, while exerting an attractive force to the goal waypoint.

$$\begin{aligned}\mathbf{A}_{i,t} = &\sum_{k=1}^{\#agents} k_{swarm}\cdot Relu\left(d_{swarm} - \|\mathbf{x}_{i,k,t}\|\right)\cdot\frac{\mathbf{x}_{i,k,t}}{\|\mathbf{x}_{i,k,t}\|} \\ &+\sum_{k=0}^{3} k_{laser}\cdot Relu\left(d'_{laser} - l_{k,t}\right)\cdot R\left(\frac{k+2}{2}\pi\right)\cdot\mathbf{i} \\ &+\frac{\mathbf{g}_{i,t}}{\|\mathbf{g}_{i,t}\|}\cdot V_{desired}\end{aligned} \tag{5}$$

In Equation 5, $\mathbf{A}_{i,t}$ is the attraction vector of agent i at time step (so not iteration) t, specifying the motion direction. Each time step the agent receives new estimates and re-computes $\mathbf{A}_{i,t}$. The first term results in repulsion away from other agents that are closer than d_{swarm}, while the second term adds repulsion from laser rangers seeing a value lower than d'_{laser}, and the third term adds attraction to the goal waypoint.

In the first term, k_{swarm} is the swarm repulsion gain, and d_{swarm} is the threshold to start avoiding agents. $\mathbf{x}_{i,k,t}$ is the vector between agent i and agent k, at time step t. The rectified linear unit ($Relu$) makes sure only close agents are repulsed.

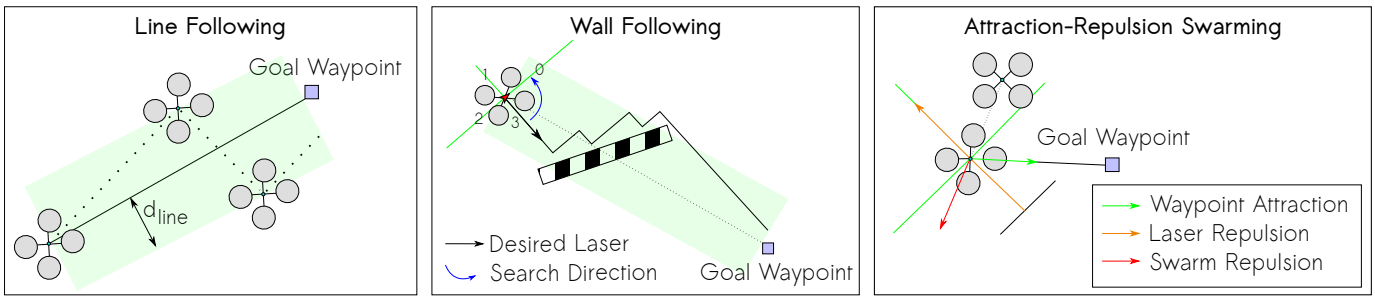


Fig. 3. Sniffy Bug's three states: line following, wall following, and attraction-repulsion swarming.

In the second term, k_{laser} is the laser repulsion gain, and d'_{laser} is the threshold to start repulsing a laser. $l_{k,t}$ is the laser reading k at step t, numbered according to Figure 3. $Relu$ makes sure only lasers recording values lower than d'_{laser} are repulsed. $R(\cdot)$ is the rotation matrix, used to rotate $\mathbf{i}$ in the direction away from laser k, such that the second term adds repulsion away from low lasers. The third term adds attraction from the agent's current position to the goal. $\mathbf{g}_{i,t}$ is the vector from agent i to the goal waypoint, at time step t. This term is scaled to be of size $V_{desired}$, which is the desired velocity, a user-defined scalar. As a last step, we normalize $\mathbf{A}_{i,t}$ to have size $V_{desired}$ too, using Equation 6.

$$\mathbf{V}_{command} = \frac{\mathbf{A}_{it}}{\|\mathbf{A}_{it}\|} \cdot V_{desired} \tag{6}$$

Here $\mathbf{V}_{command}$ is the velocity vector sent to the low-level control loops. Commanding a constant speed prevents both deadlocks in which the drones hover in place and peaks in velocity that induce collisions.

C. AutoGDM

Fully automated gas dispersion modeling based on Computational Fluid Dynamics (CFD) requires three main steps: 1) Environment generation, 2) CFD, and 3) Filament simulation (Figure 4).

1) Environment Generation: We use the procedural environment generator described in [33], which can generate environments of any desired size with a number of requested rooms. Additionally, AutoGDM allows users to insert their own 2D binary occupancy images, making it possible to model any desired 2.5D environment by extruding the 2D image.

2) CFD: CFD consists of two main stages, i.e., meshing and solving (flow field generation). We use the open-source package OpenFOAM [34] for both stages. To feed the generated environments into OpenFOAM, the binary occupancy maps are converted into 3D triangulated surface geometries of the flow volume. We detect the largest volume in the image and declare it as our flow volume and test area. We create a mesh using OpenFOAM `blockMesh` and `snappyHexMesh`, and assign inlet and outlet boundary conditions randomly to vertical surfaces. Finally, we use OpenFOAM `pimpleFOAM` to solve for kinematic pressure, p, and the velocity vector, U.

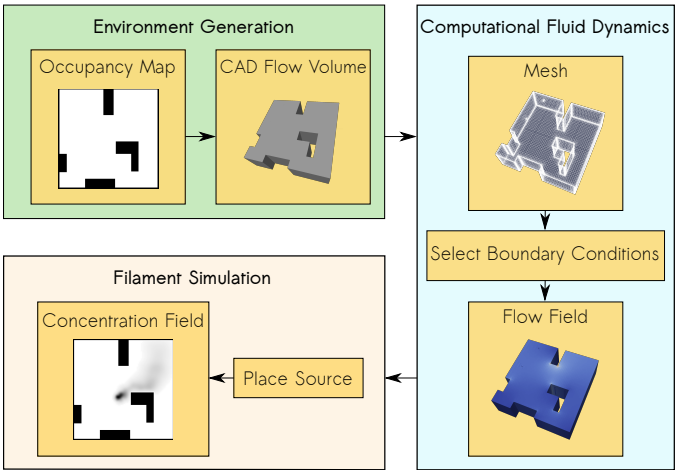


Fig. 4. AutoGDM, a fully automated environment generation and gas dispersion modeling pipeline.

3) Filament simulation: In the final stage of AutoGDM, we use the GADEN ROS package [22] to model a gas source based on filament dispersion theory. It releases filaments that expand over time, and disappear when they find their way to the outlet. The expansion of the filaments and the dispersion rate of the source (i.e., filaments dispersed per second), is random within a user-defined range.

D. Evolutionary Optimization

We feed the generated gas data into Swarmulator[1], an open source lightweight C++ simulator for simulating swarms. The agent is modelled as a point mass, which is velocity-controlled using a P controller. We describe both the environment and laser rays as a set of lines, making it extremely fast to model laser rangers. An agent 'crashes' when one of its laser rangers reads less than $0.1\,\mathrm{m}$ or when another agent is closer than $0.5\,\mathrm{m}$. The agents are fed with gas data directly from AutoGDM, which is updated every $1.0\,\mathrm{s}$ in simulation time.

Using this simulation environment, we evolve the parameters of Sniffy Bug with the 'simple genetic algorithm' from the open-source PyGMO/PAGMO package [35]. The population consists of 50 individuals and is evolved for 400 generations. The algorithm uses tournament selection, mating through exponential crossover and mutation using a polynomial mutation

[1]https://github.com/coppolam/swarmulator

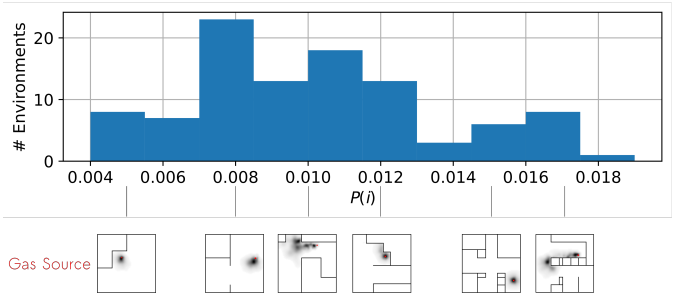


Fig. 5. Environment selection probability $P(i)$ for all 100 environments at the end of evolution. The environments below the x-axis show that harder environments, with higher $P(i)$, contain more obstacles and local optima.

operator. The mutation probability is 0.1, while the crossover probability is 0.9. The genome consists of 13 different variables, as shown in Table I, including their ranges set during evolution. Parameters that have a physical meaning when negative are allowed to be negative, while variables such as time and distance need to be positive.

Each agent's cost is defined as its average distance to source with an added penalty (+ 1.0) for a crash. Even without a penalty the agents will learn to avoid obstacles to some capacity, but a penalty stimulates prudence. Other metrics like 'time to source' were also considered, but we found average distance to work best and to be most objective. It leads to finding the most direct paths and staying close to the source.

In each generation, we select n environments out of the total of m environments generated using AutoGDM. As considerable heterogeneity exists between environments, we may end up with a controller that is only able to solve easy environments. This is known as the problem of hard instances. To tackle this problem, we study the effect of 'doping' [36] on performance in simulation. When using doping, the probability of selecting environment number i is:

$$P(i) = \frac{D(i)}{\sum_{k=0}^{m} D(k)} \tag{7}$$

$D(i)$ is the 'difficulty' of environment i, computed based on previous experience. If environment i is selected to be evaluated, we save the median of all 50 costs in the population. We use median instead of mean to avoid a small number of poor-performing agents to have a large impact. $D(i)$ is the mean of the last three recorded medians. If no history is present, $D(i)$ is the average of all difficulties of all environments. Using Equation 7 implies that we start with an even distribution, but over time give environments with greater difficulty a higher chance to be selected in each draw. When not using doping, we set $P(i) = \frac{1}{m}$, resulting in a uniform distribution.

III. Results

A. Training in Simulation

For evolution, we used AutoGDM to randomly generate 100 environments of 10×10 m in size, the size of our experimental arena. This size is sufficiently large for creating environments with separate rooms, in which local maxima of gas concentration may exist. At the same time, it is sufficiently small for exploration by a limited number of robots. We use 3 agents, with $V_{desired} = 0.5\,\mathrm{m/s}$. Figure 5 shows that the generated environments differ in obstacle configuration and gas distribution. During each generation, every genome is evaluated on a random set of 10 out of the total 100 environments, with a maximum duration per run of $100\,\mathrm{s}$. All headings and starting positions are randomized after each generation. Agents are spawned in some part of the area so that a path exists towards the gas source.

Variable	Manually Selected	Evolved	Evolution range
ω	0.5	0.271	[-5,5]
φ_p	0.8	-0.333	[-5,5]
φ_g	2.0	1.856	[-5,5]
ω'	0.3	1.571	[-5,5]
r_r	0.7	2.034	[0,5]
t_{wp}	10.0	51.979	[0,100]
d_{wp}	0.5	2.690	[0,5]
d_{laser}	1.5	1.407	[0,5]
d_{swarm}	1.5	0.782	[0,5]
d_{line}	0.2	0.469	[0,1]
k_{laser}	5.0	16.167	[0,20]
k_{swarm}	15.0	10.032	[0,20]
d'_{laser}	1.5	0.594	[0,5]

TABLE I
PARAMETERS EVOLVED IN EVOLUTION USING DOPING, CONSULT SECTION II-B FOR THE MEANING OF THE VARIABLES.

We assess training results for training with doping. Table I shows the evolved parameters in comparison with the manually designed parameters. r_{range} is set to $10\,\mathrm{m}$, creating random waypoints in a box of $10\,\mathrm{m}$ in size around the agent during 'exploring'. This box is scaled by evolved parameter r_r (Equation 3). When generating new waypoints, the agent has learned to move strongly towards the swarm's best-seen position $\mathbf{s}_j$, away from its personal best-seen position $\mathbf{p}_{i,j}$, and towards its previously computed goal $\mathbf{g}_{i,j-1}$. We expect this to be best in the environments encountered in evolution, as in most cases only one optimal position (with the highest concentration) exists. Hence, it does not need its personal best position to avoid converging to local optima. ω adds 'momentum' to the waypoints generated, increasing stability.

d_{swarm} shows that attraction-repulsion swarming is engaged only when another agent is within $0.782\,\mathrm{m}$. This is substantially lower than the manually chosen $1.5\,\mathrm{m}$, which can be explained by a lower cost when agents stay close to each other after finding the source. It also makes it easier to pass through narrow passages, but could increase the collision risk when agents are close to each other for a long time. Furthermore, d_{wp} is set to $2.69\,\mathrm{m}$, which is much higher than our manual choice and the timer has been practically disabled (t_{wp} = 51.979). Instead of using the timeout, the evolved version uses PSO to determine the desired direction to follow, until it has travelled far enough and generates a new waypoint. For obstacle avoidance we see that the manual parameters are more conservative than the evolved counterparts. Being less conservative allows the agents to get closer to the source quicker, at the cost of an increased collision risk. This is an interesting trade-off, balancing individual collision risks with more efficient gas source localization.

	Success Rate	Avg Distance to Source [m]	Avg Time to Source [s]
Manual Params PSO	90 %	3.06	51.58
Manual Params Anemotaxis	91 %	4.12	60.56
Manual Params Chemotaxis	80 %	4.31	68.61
Evolved PSO w/o Doping	89 %	2.95	44.33
Evolved PSO with Doping	**92 %**	**2.75**	**41.94**

TABLE II
SNIFFY BUG EVALUATED ON 100 RANDOMLY GENERATED ENVIRONMENTS.

After training, the probability for each environment in each draw can be evaluated as a measure of difficulty. Figure 5 shows a histogram of all 100 probabilities along with some environments on the spectrum. Generally, more cluttered environments with more local minima are more difficult.

B. Baseline Comparison in Simulation

It is difficult to compare Sniffy Bug with a baseline algorithm, since it would need to seek a gas source in cluttered environments, avoiding obstacles with only four laser rangers and navigating without external positioning. To our knowledge, such an algorithm does not yet exist. Hence, we replace the gas-seeking part of Sniffy Bug (PSO) by two other well-known strategies, leading to (i) Sniffy Bug with anemotaxis, and (ii) Sniffy Bug with chemotaxis. For anemotaxis, inspired by [7], waypoints are placed randomly when no gas is present, and upwind when gas is detected. Then, when the agent loses track of the plume, it generates random waypoints around the last point inside the plume, until it finds the plume again. Similarly, the chemotaxis baseline uses Sniffy Bug for avoidance of obstacles and other agents, but places waypoints in the direction of the gas concentration gradient if a gradient is present. The chemotaxis baseline determines the gradient by moving a small distance in x and y before computing a new waypoint, while the anemotaxis baseline receives ground-truth airflow data straight from AutoGDM. We expect that the baseline algorithms' sensor measurements - and hence performance - would transfer less well to real experiments. For chemotaxis we expect that estimating the gradient in presence of noise and drift of the sensor data will yield inaccurate measurements. Anemotaxis based on measuring the weak airflow in indoor environments will be even more challenging.

We evaluate the models in simulation on all 100 generated test environments, and randomize start position 10 times in each environment, for a total of 1,000 runs for each model. We record different performance metrics: 1) success rate, 2) average distance to source, and 3) average time to source. Success rate is defined as the fraction of runs during which at least one of the agents reaches within $1.5\,\mathrm{m}$ from the source, whereas average time to source is the average time it takes an agent to reach within $1.5\,\mathrm{m}$ from the source. For agents not reaching the source, $100\,\mathrm{s}$ is taken as time to source.

Table II shows that Sniffy Bug with PSO outperforms anemotaxis and chemotaxis in average time and distance to source, and has a success rate similar to the anemotaxis baseline. Chemotaxis suffers from the gas gradient not always pointing in the direction of the source. Due to local optima, and a lack of observable gradient further away from the source, chemotaxis is not as effective as PSO or anemotaxis. Since we expect the sim2real gap to be much more severe for anemotaxis than for PSO, which only needs gas readings, we decide to proceed with PSO.

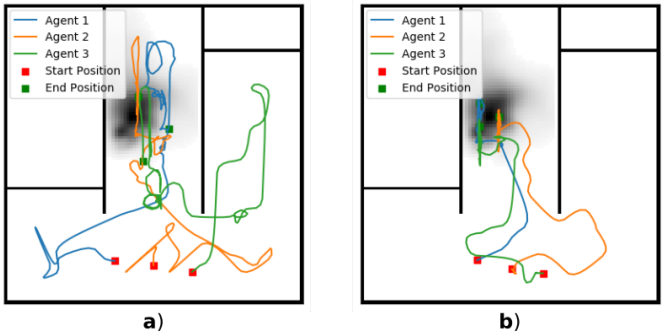


Fig. 6. Sniffy Bug using PSO evaluated, using: a) manual parameters, b) parameters evolved using doping.

C. Doping in Simulation

We now compare Sniffy Bug using PSO with manual parameters, parameters evolved without doping, and parameters evolved with doping. Table II shows that the evolved parameters without doping find the source quicker, and with a smaller average distance to source, compared to the manual parameters. However, its success rate is slightly inferior to the manual parameters. This is caused by the hard instance problem [36]: the parameters perform well on average but fail on the more difficult environments.

Figure 6 shows runs in simulation of the manual and evolved version with doping respectively, with the same initial conditions. The parameters evolved with doping outperform the other controllers in all metrics. Doping helps to tackle harder environments, improving success rate and thereby average distance to source and time to source. This effect is further exemplified by Figure 7. The doping controller does not only result in a lower average distance to source, it also shows a smaller spread. Doping managed to reduce the size of the long tail that is present in the set of evolved parameters.

As we test on a finite set of environments, and the assumptions of normality may not hold, we use the empirical bootstrap [37] to compare the means of the average distance to source. Using 1,000,000 bootstrap iterations, we find that only when comparing the manual parameters with evolved parameters with doping, the null hypothesis can be rejected, with $P = 2 \times 10^{-6}$, showing that parameters evolved with doping perform significantly better than manual parameters.

D. Flight Tests

Finally, we transfer the evolved solution to the real world, validating our methodology. We deploy our swarm in four different environments of 10×10 m in size, as shown in Figure 8. We place a small can of isopropyl alcohol with a 5V computer fan within the room as the gas source. We compare manual parameters with the parameters evolved using doping, by comparing their recorded gas readings. Each set

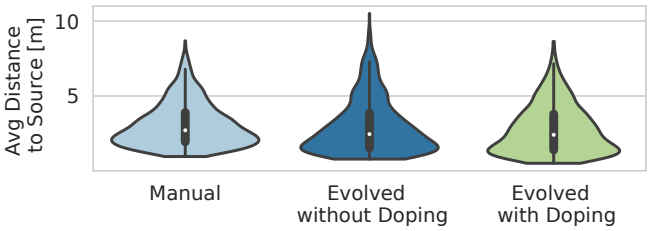
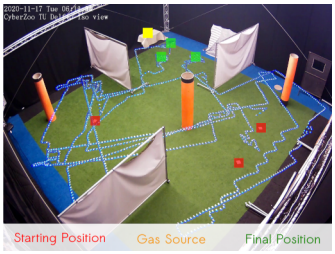
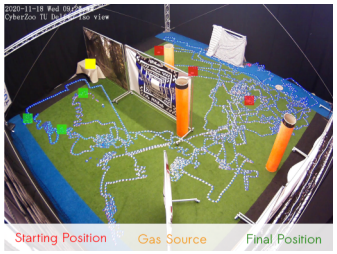


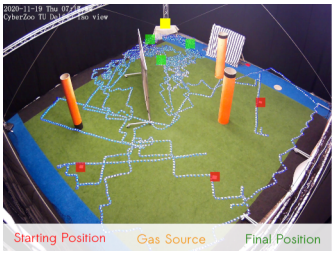
Fig. 7. Three sets of parameters evaluated in simulation for Sniffy Bug using PSO: 1) manually set parameters, 2) parameters evolved without doping, and 3) parameters evolved using doping [36] to address the hard instance problem.



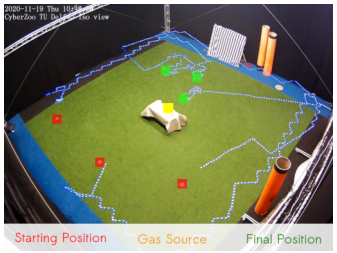
(a) Environment 1: the agents need to explore the 'room' in the upper corner to smell the gas.



(b) Environment 2: the agents need to go around a long wall to locate the source.



(c) Environment 3: an easier environment with some obstacles.



(d) Environment 4: an obstacle-free environment.

Fig. 8. Time-lapse images of real-world experiments in four distinct environment setups, 10×10 m in size, seeking a real isopropyl alcohol source. The nano quadcopters' trajectories are visible due to their blue lights.

of parameters is evaluated three times for each environment, resulting in a total of 24 runs. A run is terminated when: 1) the swarm is stationary and close to the source, or 2) at least two members of the swarm crash. Each run lasts at least 120 s. Figure 9 corresponds to the run depicted in Figure 8a.

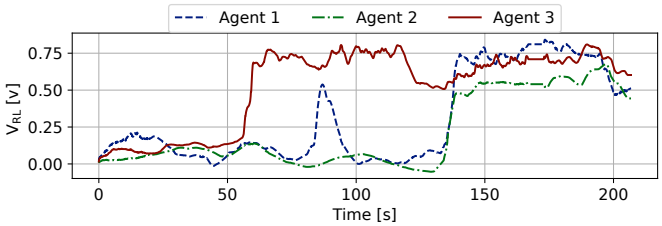
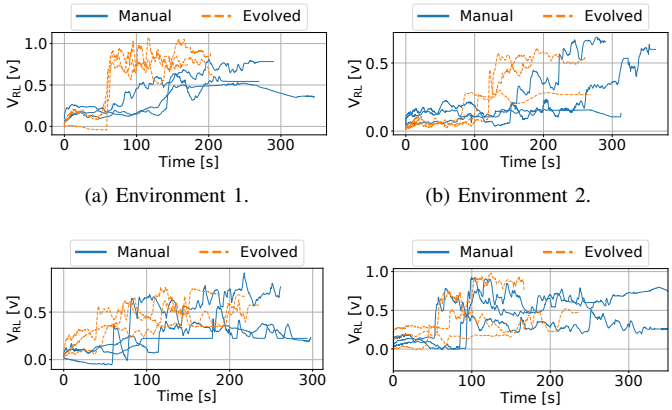


Fig. 9. Evolved Sniffy Bug seeking an isopropyl alcohol gas source in environment 1. After Agent 3 finds the source, all three agents quickly find their way to higher concentrations.

Figure 10 shows the maximum recorded gas readings by the swarm, for each time step for each run. Especially for Environment 1, it clearly shows the more efficient source



(a) Environment 1.

(b) Environment 2.

(c) Environment 3.

(d) Environment 4.

Fig. 10. Maximum recorded gas reading by the swarm, for each time step for each run.

	Manual		**Evolved**	
	Avg ± std	Max ± std	Avg ± std	Max ± std
Env 1	0.250 ± 0.036	0.406 ± 0.049	**0.330** ± 0.046	0.566 ± 0.069
Env 2	0.162 ± 0.055	0.214 ± 0.070	**0.165** ± 0.046	0.237 ± 0.063
Env 3	0.200 ± 0.074	0.300 ± 0.103	**0.258** ± 0.045	0.412 ± 0.029
Env 4	**0.240** ± 0.123	0.398 ± 0.143	0.176 ± 0.062	0.349 ± 0.151

TABLE III
AVERAGE AND MAXIMUM SMELLED CONCENTRATION BY THE SWARM, FOR MANUAL AND EVOLVED PARAMETERS, AVERAGED OVER 3 RUNS.

seeking behavior of our evolved controller. Table III shows the average and maximum observed concentrations by the swarm, averaged over all three runs per environment. It shows that for environments with obstacles, our evolved controller outperforms the manual controller in average observed gas readings and average maximum observed gas readings.

The evolved controller was able to reach the source within ± 2.0 m in 11 out of 12 runs, with one failed run due to converging towards a local optimum in environment 4 (possibly due to sensor drift). The manual parameters failed once in environment 2 and once in environment 3. The manual parameters were less safe around obstacles, recording a total of 3 crashes in environments 1 and 2. The evolved parameters recorded only one obstacle crash in all runs (environment 2). We hypothesize that the more efficient, evolved GSL strategy reduces the time around dangerous obstacles.

On the other hand, the evolved parameters recorded 2 drone crashes, both in environment 4, when the agents were really close to each other and the source for extended periods of time. The manual parameters result in more distance between agents, making it more robust against downwash and momentarily poor relative position estimates. This can be avoided in future work by a higher penalty for collisions during evolution, or classifying a run as a crash when agents are, for instance, 0.8 m away from each other instead of 0.5 m.

The results show that AutoGDM can be used to evolve a controller that not only works in challenging real-world conditions, but even outperforms manually chosen parameters. While GADEN [22] and OpenFOAM [34] are by themselves already validated packages, the results corroborate the validity of our simulation pipeline, from the environment generation to the simulated drones' particle motion model.

IV. Conclusion

We have introduced a novel bug algorithm, Sniffy Bug, leading to the first fully autonomous swarm of gas-seeking nano quadcopters. We evolved the parameters of the algorithm in simulation and successfully transferred the solution to challenging real-world environments. The evolved parameters outperformed a human-designed controller in all metrics, both in simulation and robot experiments. We also contribute the first fully-automated environment generation and gas dispersion modeling pipeline, AutoGDM, that allows for learning GSL in simulated complex environments.

In future work, our methodology may be extended to larger swarms of nano quadcopters, exploring buildings and seeking a gas source fully autonomously. PSO was designed to work in large optimization problems with many local optima, and is likely to extend to more complex configurations and to GSL in 3D. Finally, we hope that our approach can serve as inspiration for tackling also other complex tasks with swarms of resource-constrained nano quadcopters.

References

[1] M. Coppola, K. N. McGuire, C. De Wagter, and G. C. H. E. de Croon, "A survey on swarming with micro air vehicles: Fundamental challenges and constraints," *Frontiers in Robotics and AI*, vol. 7, p. 18, 2020.

[2] H. Xu, L. Wang, Y. Zhang, K. Qiu, and S. Shen, "Decentralized visual-inertial-uwb fusion for relative state estimation of aerial swarm," *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[3] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[4] B. Bodin, H. Wagstaff, S. Saecdi, L. Nardi, E. Vespa, J. Mawer, A. Nisbet, M. Lujan, S. Furber, A. J. Davison, P. H. J. Kelly, and M. F. P. O'Boyle, "SLAMBench2: Multi-objective head-to-head benchmarking for visual slam," in *ICRA 2018*, 2018, pp. 3637–3644.

[5] E. Tang, S. Niknam, and T. Stefanov, "Enabling cognitive autonomy on small drones by efficient on-board embedded computing: An ORB-SLAM2 case study," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*, 2019, pp. 108–115.

[6] K. N. McGuire, C. De Wagter, K. Tuyls, H. J. Kappen, and G. C. H. E. de Croon, "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment," *Science Robotics*, vol. 4, no. 35, 2019.

[7] M. J. Anderson, J. G. Sullivan, T. Horiuchi, S. B. Fuller, and T. L. Daniel, "A bio-hybrid odor-guided autonomous palm-sized air vehicle," *Bioinspiration & Biomimetics*, 2020.

[8] C. Ercolani and A. Martinoli, "3d odor source localization using a micro aerial vehicle: System design and performance evaluation," in *IROS 2020*, 2020, pp. 6194–6200.

[9] C. Song, Y. He, B. Ristic, and X. Lei, "Collaborative infotaxis: Searching for a signal-emitting source based on particle filter and Gaussian fitting," *Robotics and Autonomous Systems*, vol. 125, p. 103414, 2020.

[10] M. Vergassola, E. Villermaux, and B. I. Shraiman, "'Infotaxis' as a strategy for searching without gradients," *Nature*, vol. 445, no. 7126, pp. 406–409, 2007.

[11] Y. Kuwana, S. Nagasawa, I. Shimoyama, and R. Kanzaki, "Synthesis of the pheromone-oriented behaviour of silkworm moths by a mobile robot with moth antennae as pheromone sensors." *Biosensors and Bioelectronics*, vol. 14, no. 2, pp. 195 – 202, 1999.

[12] J. Adler, "The sensing of chemicals by bacteria," *Scientific American*, vol. 234, no. 4, pp. 40–47, 1976.

[13] T. Lochmatter and A. Martinoli, *Understanding the Potential Impact of Multiple Robots in Odor Source Localization*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 239–250.

[14] R. D. Beer and J. C. Gallagher, *Evolving Dynamical Neural Networks for Adaptive Behavior*, 1992, vol. 1, no. 1.

[15] E. J. Izquierdo and T. Buhrmann, "Analysis of a dynamical recurrent neural network evolved for two qualitatively different tasks: Walking and chemotaxis," *ALIFE 2008*, pp. 257–264, 2008.

[16] E. J. Izquierdo and S. R. Lockery, "Evolution and analysis of minimal neural circuits for klinotaxis in Caenorhabditis elegans," *Journal of Neuroscience*, vol. 30, no. 39, pp. 12 908–12 917, 2010.

[17] G. de Croon, L. O'Connor, C. Nicol, and D. Izzo, "Evolutionary robotics approach to odor source localization," *Neurocomputing*, vol. 121, no. December, pp. 481–497, 2013.

[18] J. L. Wei, Q. H. Meng, C. Yan, M. Zeng, and W. Li, "Multi-Robot gas-source localization based on reinforcement learning," *2012 IEEE International Conference on Robotics and Biomimetics, ROBIO 2012 - Conference Digest*, pp. 1440–1445, 2012.

[19] Y. Kuwana, I. Shimoyama, Y. Sayama, and H. Miura, "Synthesis of pheromone-oriented emergent behavior of a silkworm moth," *IEEE International Conference on Intelligent Robots and Systems*, vol. 3, pp. 1722–1729, 1996.

[20] A. T. Hayes, A. Martinoli, and R. M. Goodman, "Swarm robotic odor localization: Off-line optimization and validation with real robots," *Robotica*, vol. 21, no. 4, p. 427–441, 2003.

[21] B. P. Duisterhof, S. Krishnan, J. J. Cruz, C. R. Banbury, W. Fu, A. Faust, G. C. H. E. de Croon, and V. J. Reddi, "Tiny robot learning (tinyrl) for source seeking on a nano quadcopter," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[22] J. Monroy, V. Hernandez-Bennetts, H. Fan, A. Lilienthal, and J. Gonzalez-Jimenez, "GADEN: A 3D gas dispersion simulator for mobile robot olfaction in realistic environments," *MDPI Sensors*, vol. 17, no. 7: 1479, pp. 1–16, 2017.

[23] G. Cabrita, P. Sousa, and L. Marques, "Player/stage simulation of olfactory experiments," 11 2010, pp. 1120 – 1125.

[24] X. Chen and J. Huang, "Combining particle filter algorithm with bio-inspired anemotaxis behavior: A smoke plume tracking method and its robotic experiment validation," *Measurement*, vol. 154, p. 107482, 2020.

[25] M. Asenov, M. Rutkauskas, D. Reid, K. Subr, and S. Ramamoorthy, "Active localization of gas leaks using fluid simulation," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1776–1783, 2019.

[26] E. M. Moraud and D. Martinez, "Effectiveness and robustness of robot infotaxis for searching in dilute conditions," *Frontiers in Neurorobotics*, vol. 4, no. MAR, pp. 1–8, 2010.

[27] N. Voges, A. Chaffiol, P. Lucas, and D. Martinez, "Reactive searching and infotaxis in odor source localization," *PLOS Computational Biology*, vol. 10, no. 10, pp. 1–13, 10 2014.

[28] W. Jatmiko, K. Sekiyama, and T. Fukuda, "A PSO-based mobile robot for odor source localization in dynamic advection-diffusion with obstacles environment: theory, simulation and measurement," *IEEE Computational Intelligence Magazine*, vol. 2, no. 2, pp. 37–51, 2007.

[29] J. A. Steiner, J. R. Bourne, X. He, D. M. Cropek, and K. K. Leang, "Chemical-source localization using a swarm of decentralized unmanned aerial vehicles for urban/suburban environments," *ASME Dynamic Systems and Control Conference, DSCC 2019*, vol. 3, 2019.

[30] S. Li, M. Coppola, C. D. Wagter, and G. C. H. E. de Croon, "An autonomous swarm of micro flying robots with range-based relative localization," https://arxiv.org/abs/2003.05853, 2020.

[31] W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński, and P. Kozierski, "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering," in *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2017, pp. 37–42.

[32] J. Burgués, V. Hernández, A. J. Lilienthal, and S. Marco, "Smelling nano aerial vehicle for gas source localization and mapping," *Sensors (Switzerland)*, vol. 19, no. 3, 2019.

[33] K. McGuire, G. de Croon, and K. Tuyls, "A comparative study of bug algorithms for robot navigation," *Robotics and Autonomous Systems*, vol. 121, p. 103261, 2019.

[34] H. Jasak, "OpenFOAM: Open source CFD in research and industry," *International Journal of Naval Architecture and Ocean Engineering*, vol. 1, no. 2, pp. 89 – 94, 2009.

[35] D. Izzo, M. Ruciński, and F. Biscani, *The Generalized Island Model*, 01 2012, vol. 415, pp. 151–169.

[36] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma, "DECA: The doping-driven evolutionary control algorithm," *Applied Artificial Intelligence*, vol. 22, pp. 169–197, 03 2008.

[37] B. Efron, "Bootstrap methods: Another look at the jackknife," *Ann. Statist.*, vol. 7, no. 1, pp. 1–26, 01 1979.