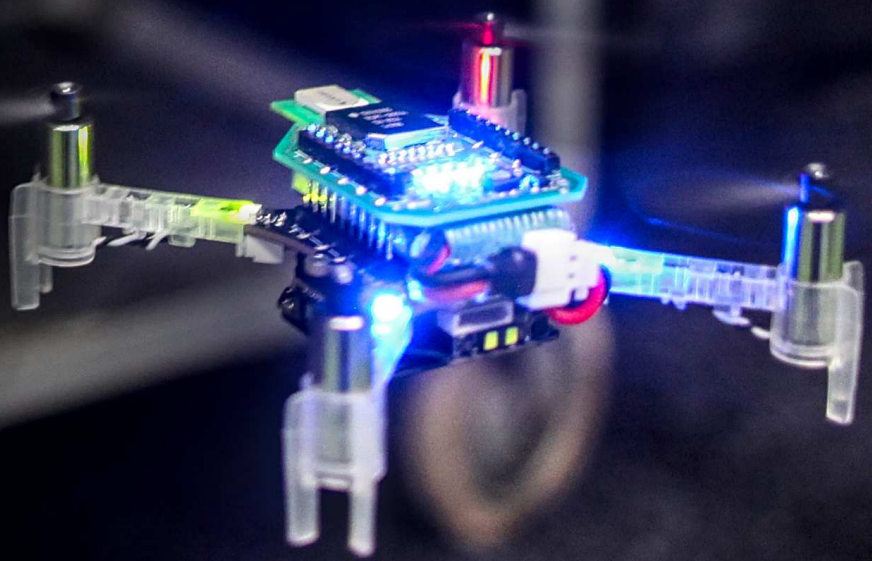
Sniffy Bug

A swarm of gas-seeking nano quadcopters

Bardienus Pieter Duisterhof

Technische Universiteit Delft



TU Delft

Sniffy Bug

A swarm of gas-seeking nano quadcopters

by

Bardienus Pieter Duisterhof

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday December 15, 2020 at 9:30 AM.

Student number:	4442695	
Project duration:	January 1, 2020 – December 15, 2020	
Thesis committee:	Prof. dr. ir. G. C. H. E. de Croon,	TU Delft, MAVLab
	Dr. ir. C. J. M. Verhoeven,	TU Delft, EWI
	Ir. C. de Wagter,	TU Delft, MAVLab

This thesis is confidential and cannot be made public until May 1, 2020.

An electronic version of this thesis will be available at `http://repository.tudelft.nl/`.

TU Delft

Preface

After a year of full-time research in the MAVLab, this work wraps up my time at TU Delft. This project is the next step in a collaboration between the Edge Computing Lab at Harvard, and the MAVLab at TU Delft. The first project at Harvard enabled phototaxis by running a deep-rl policy onboard a CrazyFlie. In this project we decided to take it a step further, by solving a much harder problem with a collaborative and fully autonomous swarm.

This document consists of two parts: first, a scientific article that describes our methodology and results. We are planning to submit this article, likely with some additional improvements, to IROS in the upcoming year. Next, a thorough literature review is attached, which was carried out in the early stages of the project. It reviews related work not only on the topic of gas source localization, but also details previous contributions in gas source modelling and learning methods.

This project, my master thesis, has been a great learning experience. It is the first time I developed the full stack: hardware, simulation/GDM pipeline, and algorithm. It has certainly helped me to become a better robotics engineer and researcher, and has convinced me to pursue a career in science.

Many people around me contributed to this project. First and foremost, Guido started guiding me long before this project. Since 2016, Guido has spent tremendous time helping me to become a better robotics researcher, by spending many hours explaining me different concepts in computer vision and AI. Most importantly, he has made me realize that I really love science and want to pursue a career in this direction. Without him my career path probably would have looked much different. Other people in the MAVLab that have helped tremendously over the years include (but is not limited to): Kimberly, Matej, Diana, Shushuai, Freek, Erik, Kirk, Tom, Jesse, Julien, Christophe, Bart, Mario, Fede, Nilay, Ewoud and Sven. On a personal note, I would like to thank all of my friends, family and Paulien for their continued support. I could not have done it without them.

Bardienus Pieter Duisterhof
Delft, December 2020

Contents

Scientific Article

Sniffy Bug: A fully autonomous and collaborative swarm of gas-seeking nano quadcopters in cluttered environments

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

***Abstract*—Nano quadcopters are ideal for gas source localization (GSL) as they are cheap, safe and agile. However, previous algorithms are unsuitable for nano quadcopters, as they rely on heavy sensors, require too large computational resources, or only solve simple scenarios without obstacles. In this work, we propose a novel bug algorithm named 'Sniffy Bug', that allows a swarm of gas-seeking nano quadcopters to localize a gas source in an unknown, cluttered and GPS-denied environment. Sniffy Bug is capable of efficient GSL with extremely little sensory input and computational resources, operating within the strict resource constraints of a nano quadcopter. The algorithm foresees in the avoidance of obstacles and other swarm members, while pursuing desired waypoints. The waypoints are set for exploration, and, when a single swarm member has sensed the gas, by a particle swarm optimization-based procedure. We evolve all the parameters of the bug (and PSO) algorithm, with a novel automated end-to-end simulation and benchmark platform, AutoGDM. This platform enables fully automated end-to-end environment generation and gas dispersion modelling (GDM), not only allowing for learning in simulation but also providing the first GSL benchmark. We show that evolved Sniffy Bug outperforms manually selected parameters in challenging, cluttered environments in the real world. To this end, we show that a lightweight and mapless bug algorithm can be evolved to complete a complex task, and enable the first fully autonomous swarm of collaborative gas-seeking nano quadcopters.**

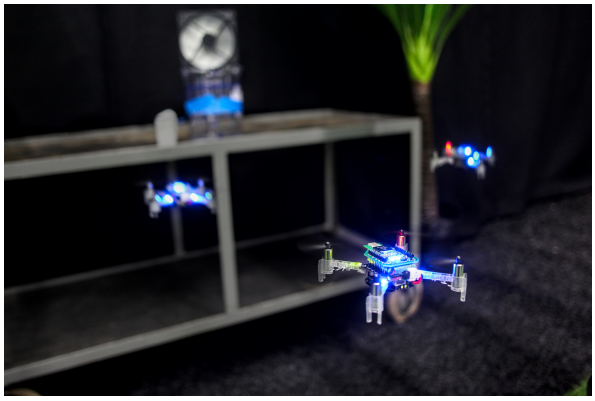


Fig. 1. A fully autonomous and collaborative swarm of gas-seeking nano quadcopters, seeking an alcohol source. The source is visible in the background: a spinning fan above a can of alcohol.

I. Introduction

Gas source localization (GSL) is an important application for search and rescue, inspection, and other jobs that are too dangerous or time-consuming for humans. Imagine cheap and disposable aerial robots inspecting ship hauls for leaks, aiding search for gas leaks in chemical plants, or quickly localizing a dangerous gas leak in a building. For that reality, we need small, agile, inexpensive and safe robots capable of complex tasks in GPS-denied environments. Therefore, a swarm of nano quadcopters is an ideal candidate for GSL.

To enable a gas-seeking swarm, the nano quadcopters need to navigate in unknown, cluttered and 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) [10, 39]. State-of-the-art methods use heavy sensors like LiDAR and high-resolution cameras to construct a map of the environment, while also estimating the robot's position (e.g., Orb-slam2 [27]). While robust, they require large computational resources to construct and store maps of the environment, making it impossible to run them onboard nano quadcopters. The most promising solution for navigation of a swarm of nano quadcopters has been obtained in [23], which introduced a novel bug algorithm that allowed a swarm of nano quadcopters to explore unknown environments and come back to the starting location.

In addition to navigation in unknown environments, a swarm of gas-seeking nano quadcopters needs a robust strategy to efficiently locate the source, which by itself is already a highly challenging task. A major factor contributing to this challenge is the currently available poor sensor quality compared to animals' smelling capabilities [2], and the dynamic nature of the problem. As a result, most works focused on simple environments without obstacle using a single robot [8, 3, 26, 34], drastically lowering difficulty of navigation and gas seeking (e.g., fewer local optima).

The currently available probabilistic GSL strategies [30, 33] require too large computational resources and do not take obstacles into account in their plume models. Bio-inspired finite-state machines [20, 1, 9] have very low computational requirements, but do not easily extend to a multi-agent setup or more complex environments with obstacles [22]. Most promising in this area is the use of particle swarm optimization (PSO) for GSL, as it may be able to deal with the local optima in gas concentration that arise in more complex environments. Closest to our work is an implementation of PSO on a group of large, outdoor flying quadcopters [32], 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. Our algorithm makes the nano quadcopters collaborate in finding the source by estimating relative positions using Ultra-Wideband (UWB), and by communicating observed gas concentrations. 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 modelling (GDM) pipeline, which we term 'AutoGDM' (Figure 4). Previous works use disparate and often simplified gas models, making it hard to compare different algorithms. AutoGDM allows for better comparison and learning in simulation, thus forming a first GSL benchmark. We validate our approach, and therefore AutoGDM, by robot demonstrations in cluttered environments, showing evolved parameters outperform manual

[1]Student, [2]Supervisor, [3]Collaborator

parameters.

In summary, we contribute:

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

In the remainder of this article, we review related work (Section II), lay down our methodology (Section III), present simulation and flight results (Section IV), and draw conclusions (Section V).

II. Related Work

Previous work in the area of GSL proposed novel algorithms, contributed software tools for the community, and presented new robot designs. We now review the state of the art in these areas separately.

The existing algorithms for GSL generally belong to one or more of the following categories: 1) bio-inspired algorithms, 2) probabilistic algorithms, 3) multi-agent algorithms, and 4) learning-based algorithms. Bio-inspired algorithms draw inspiration from organisms such as silkworm moths [20], E. coli bacteria [1] or dung beetles [9]. The earliest GSL strategies performed gradient-following for source localization, inspired by small-scale animals that live in viscous environments. For example, an E. coli-inspired algorithm [1] determines its next heading based on chemical gradient only. Gradient-based strategies fail in larger-scale environments [38] that are typical of gas source localization in air, where wind and turbulence can cause multiple local maxima in gas distribution. For such environments, inspiration has been drawn from flying insects, such as moths [20], as they do not just use chemical measurements, but also wind-vector information. Even though they perform relatively well, they are not designed for complex multi-room environments and do not easily extend to a multi-agent setup.

Probabilistic strategies were designed to be more robust in turbulent conditions, though it remains unclear if that is true in the real world [34]. Infotaxis [33] constructs a 'belief map', keeping track of spatial probability of source presence, based on a history of chemical and wind readings. It does so by tracing back 'peaks' in concentration, based on local wind and gas readings. After infotaxis was introduced in 2007, the community quickly adopted it, introducing more probabilistic methods and implementations on robots [8]. Source term estimation (STE) [13] is a popular subset of probabilistic algorithms, fitting measurements to a pre-defined model usually employing some form of Bayesian inference. The primary challenge for implementation on a nano quadcopter is the high computational cost and memory requirement. Additionally, probabilistic algorithms use simplified plume models, giving poor predictions in complex environments. Probabilistic methods have not yet considered obstacles in their plume models, which may reduce the accuracy of their predictions. It would require a (partial) map of the environment, further increasing complexity.

Multi-agent approaches are useful when deployed in emergency response, as robustness and efficiency are key to mission success. Bio-inspired algorithms can be deployed using multiple robots. Especially the moth-inspired spiral-surge algorithm has been studied in a multi-agent context, though its lack of collaboration results in an inefficient strategy [22]. Formation-based swarming [7] was attempted for GSL, leveraging the gradient between agents in a formation, adding attraction towards agents sensing higher concentrations. A downside of formation swarming is its suboptimal exploration (i.e., agents stay together), making it unlikely to be effective in larger and more complex environments.

Particle swarm optimization (PSO) [18] represents a promising algorithm for locating gas sources in complex environments, as it is a population-based optimization algorithm that can deal, to a certain extent, with local optima. Its property of a heuristic search method (requiring only a value and no gradient measurement) and its population-based nature make it a good fit for swarm-based GSL. In fact, it was deployed onboard a swarm of large gas-seeking quadcopters in [32], using LiDAR to avoid obstacles and GPS for positioning. We extend on this work, by deploying a novel bug algorithm, inspired by [23], that allows for autonomous and collaborative GSL on nano quadcopters in complex, cluttered, GPS-denied environments.

Learning-based strategies have been attempted for GSL, but the work in this direction is relatively sparse. Reinforcement and evolutionary learning approaches have been first attempted in [4, 19], and later in [14, 15, 11, 35]. The results in simulation look very promising, but real-world testing was limited to very small areas ($< 1 \times 1$ m) without obstacles [19]. We contribute the first transfer of a learning-based approach to GSL, from simulation to a swarm of nano quadcopters in challenging environments in the real world.

To allow for successful deployment of learning-based solutions in the real world, we need to train agents on a large number of environments with varying conditions and configurations. So far, gas dispersion modelling has been a time-intensive process, requiring domain knowledge for accurate modelling. As a consequence, most studies on GSL are based on a single or only a few environments with gas distributions over time [36]. Moreover, to the best of our knowledge, only very limited environments have been made available to the public [25]. We believe that the field would be helped substantially by a fully automated environment generation and gas distribution modelling pipeline that is able to generate a large number of environments that vary in complexity, both in terms of the obstacles in the environment and the gas flow field. The GADEN ROS package [25] is a great contribution, automating filament simulation, but it lacks automation of environment generation, computational fluid dynamics (CFD), and source placement. We build on GADEN to propose AutoGDM, a fully automated GDM pipeline that does not only allow for training GSL agents, but also forms a first GSL benchmark.

Finally, from a hardware perspective, GSL is usually carried out on ground robots [8, 5], with most works in the hardware space focusing on improved sensing. Artificial gas sensors are highly constrained, leading researchers to design bio-hybrid systems using insect antennae [2, 29]. GSL onboard a nano quadcopter has been attempted before [6, 28, 2], but always with a single robot relying on a ground station. Fully autonomous flight of a gas-seeking nano quadcopter has not yet been achieved, due to the very strict resource constraints. We present the first fully autonomous and collaborative swarm of gas-seeking nano quadcopters, by a custom PCB for gas seeking and relative localization.

III. Method

We present our application-specific systems design (Section III-A), including a custom gas and relative localization deck. Next, we describe our novel bug algorithm (Section III-B), capable of fully autonomous GSL, while collaborating with other agents. Finally, we lay down our fully

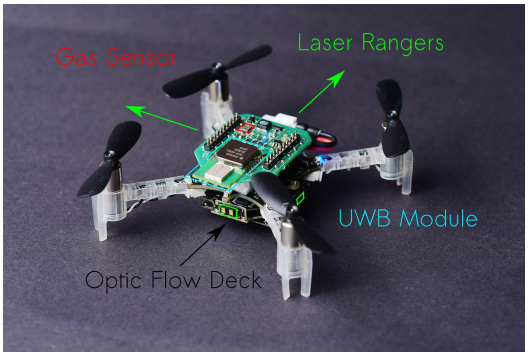


Fig. 2. A 37.5 g nano quadcopter, capable of fully autonomous waypoint tracking, obstacle avoidance, relative localization, communication and gas sensing.

automated GDM pipeline, AutoGDM (Section III-C), which we use to evolve our controller (Section III-D).

A. System Design

Our 37.5 g nano quadcopter (Figure 2), is equipped with sensors to allow for waypoint tracking, obstacle avoidance, relative localization, communication and gas sensing. For waypoint tracking, we use the BitCraze Flow deck and IMU sensors, to estimate the drone's state and track waypoints in the body frame. Additionally, we deploy the BitCraze multiranger deck, carrying four laser range sensors in the drones positive and negative x and y axis, to sense and avoid obstacles.

Finally, we design and open-source a custom PCB, capable of gas sensing and relative localization. We use a Figaro TGS8100 MEMS gas sensor, which is lightweight, cheap, and was previously deployed onboard a nano quadcopter [6]. We use it so seek alcohol, but it can be used with many other substances, such as carbon monoxide. The TGS8100 changes resistance based on exposure to gas, which can be computed according to Equation 1.

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

Here R_s is the sensor resistance, V_c circuit voltage (3.0 V), V_{RL} the voltage drop over the load resistor in a voltage divider, and R_L is the load resistor's resistance. 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 localization, we use a Decawave DWM1000 UWB module to sense relative range between agents. An extended Kalman filter (EKF) uses onboard sensing from all agents, measuring velocity, yaw rate, and height, which is fused with the observed ranges from the UWB module. It does so without relying on any external systems or magnetometers, which are known to drift [37], especially at this scale. Additionally, all agents are programmed to maintain constant yaw, as it further improves the stability and accuracy of the estimates. For a more in-depth description of the relative ranging pipeline, the reader is invited to consult [21] that proposes the above-mentioned methodology.

B. Algorithm Design

We design an algorithm that is capable of GSL using extremely little resources, both from a computational and sensor perspective. We generate waypoints in the reference frame of each agent using particle swarm optimization (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 bias. 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 using PSO, which is configured in either of two modes: '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, Equation 3 describes computation of $\mathbf{v}_{i,j}$.

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

Here $\mathbf{v}_{i,j}$ is the velocity vector of agent i in iteration j, $\mathbf{g}_{i,j-1}$ 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.

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

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.

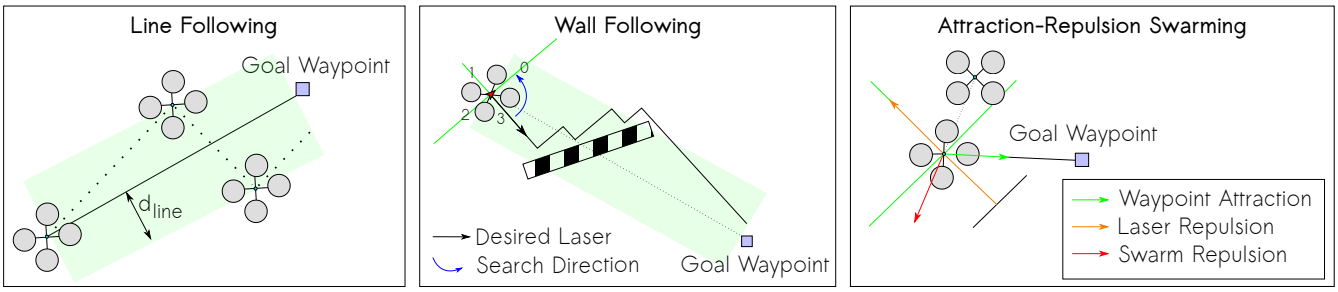


Fig. 3. Sniffy Bug's three states: line following, wall following, and attraction-repulsion swarming.

2) Waypoint tracking: Tracking these waypoints is challenging, considering that our robots only have access to 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, creating a motion 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 [24]. The wall following stage is visible in Figure 3, and laid down in detail in Algorithm 1, Appendix A. 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.

During wall following Sniffy Bug outputs '$local\_laser$', an integer between 0 and 3, indicating if the robot will move in one of four directions in the body frame ($+x, +y, -x, -y$, ENU, respectively). We use Figure 3 as an example to explain wall following in Sniffy Bug.

The agent starts by computing $desired\_laser$, 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 (below 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 detects it has avoided the obstacle, by exiting and re-entering the green zone, while it has gotten closer to the goal waypoint (as determined with odometry).

The example shown is a simple case, but Sniffy Bug is capable of avoiding more complex objects too. It will keep increasing $local\_laser$ until it finds a safe direction. To avoid oscillations, we only consider directions from $max\_reached\_laser - 1$ for anti-clockwise searching, and $max\_reached\_laser + 1$ for clockwise searching. This constraint is only imposed if the agent has considered directions beyond $desired\_laser$. In Figure 3 this means if the agent has travelled in the direction of laser 2, it will only consider laser 1 and 2. If those two lasers both see a value below d_laser, the wall follower is reset. This means we start looking from $desired\_laser$ again, but now clockwise. Using this strategy, Sniffy Bug can move around larger obstacles with complex shapes in cluttered environments.

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 none of the other agents 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.

$$\begin{aligned}\mathbf{A}_{i,j} = \sum_{k=0}^{\#agents} & k_{swarm} \cdot Relu\left(d_{swarm} - \|\mathbf{x}_{i,j} - \mathbf{x}_{k,j}\|\right) \cdot \frac{\mathbf{x}_{i,j} - \mathbf{x}_{k,j}}{\|\mathbf{x}_{i,j} - \mathbf{x}_{k,j}\|} \\ & + \sum_{k=0}^{3} k_{laser} \cdot Relu\left(d'_{laser} - l_k\right) \cdot R\left(\frac{k+2}{2}\pi\right) \cdot \mathbf{i} \\ & + \frac{\mathbf{g}_{i,j} - \mathbf{x}_{i,j}}{\|\mathbf{g}_{i,j} - \mathbf{x}_{i,j}\|} \cdot V_{desired}\end{aligned} \tag{5}$$

In Equation 5, $\mathbf{A}_{i,j}$ is the attraction vector of agent i in time step (so not iteration) j, specifying the direction the agent will move towards. Each time step the agent receives new estimates and re-computes $\mathbf{A}_{i,j}$. 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. The repulsive terms increase linear with distance, allowing it to manage multiple repulsive forces, like multiple agents passing close to a wall and each other.

k_{swarm} is the swarm repulsion gain, and d_{swarm} is the threshold to start avoiding agents. $\mathbf{x}_{k,j}$ is the position of agent k, at time step j, in the reference frame of agent i. $\mathbf{x}_{i,j}$ is agent i's own position at time step j, in its own reference frame. The rectified linear unit ($Relu$) makes sure only close agents are repulsed.

In the second term, k_{laser} is the laser repulsion gain, and d'_{laser} is the threshold to start repulsing a laser. l_k is the read of laser k, 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.

Finally, the third and final term adds attraction from the agent's current position $\mathbf{x}_{i,j}$ to the goal $\mathbf{g}_{i,j}$, which is the goal waypoint agent i is tracking in time step j. This term is scaled to be of size $V_{desired}$.

As a last step, we normalize $\mathbf{A}_{i,j}$ to have size $V_{desired}$ too, using Equation 6.

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

Here $V_{command}$ is the velocity vector we command to the low-level control loops. Commanding a constant velocity

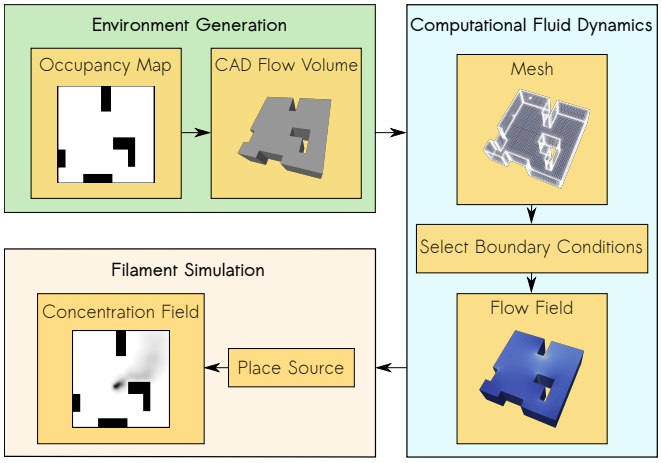


Fig. 4. AutoGDM, a fully automated environment generation and gas dispersion modelling (GDM) platform.

vector size prevents deadlocks in which the drones hover in place. Moreover, it prevents peaks in velocity that increase the probability of collisions.

C. AutoGDM

In a future with effective deployment of nano quadcopters, their behavior needs to be robust in a variety of conditions, making it possible to instantly deploy them in an unknown environment. In the case of GSL, this means our swarm needs to be evaluated in different flow conditions and environment settings, as both have great impact on task difficulty and optimal strategy. Usually manually tuning the hyperparameters of bug algorithms is a time-intensive yet critical task, almost certainly yielding a non-optimal controller. Instead, we evolve the parameters of our algorithm to reach optimal behavior in 100 randomly generated environments, by dispersing gas in virtual environments.

Gas dispersion modelling (GDM) has so far been a complicated and time-intensive task, that requires domain knowledge for accurate modelling. The GADEN ROS package [25] provides a useful tool for dispersion modelling, but does not automate the CFD-based flow field generation, source placement and environment generation. Manually creating a large number of environments is impractical, whereas a fully automated pipeline will create environments more effortlessly and consistently.

While many efforts towards more efficient GDM exist, most scientists agree that modelling based on computational fluid dynamics (CFD), is most accurate. We first model a flow field using CFD, which we then use to disperse gas 'filaments'. Hence, fully automated CFD-based GDM requires three main steps, being: 1) environment generation, 2) CFD, and 3) Filament simulation. We now discuss the different building blocks of AutoGDM separately.

1) Environment Generation: We use the procedural environment generator described in [24], that 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: 1) meshing, and 2) solving (i.e., flow field generation). We use OpenFOAM [17], an open-source CFD toolbox, to complete 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 use OpenCV to detect the largest volume in the image, and declare it as our flow volume and test area. We first create a background mesh of hexahedral cells that fills the entire volume, using OpenFOAM `blockMesh`. We then use `snappyHexMesh` to complete the mesh, creating more cells in close proximity to edges where higher gradients of flow velocities are expected.

Next, we need to define the boundary conditions, i.e., wall, inlet and outlet. Out of the n largest vertical surfaces, we randomly choose two surfaces that will be defined as inlet and outlet, while all other surfaces are defined as wall. n is a user-defined integer.

Finally, we use `pimpleFOAM` to solve for kinematic pressure, p, and the velocity vector, U.

3) Filament simulation: In the final stage of AutoGDM, we use the GADEN ROS package [25] 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], a lightweight C++ simulator for simulating swarms. The agent is modelled as a point mass, that 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, as compared to ray-tracing. An agent is deemed crashed 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.

Swarmulator spawns several threads on the CPU in parallel, evaluating multiple environments and agents concurrently, accelerating evolution. We serialize all gas data into a gas class before starting experiments, making it possible to efficiently load gas data for each environment. Compared to using the txt files generated by GADEN, our approach delivers an additional $6\times$ performance improvement for each evaluation.

Using this simulation environment, we evolve the parameters of Sniffy Bug with the 'simple genetic algorithm' from the open-source PyGMO/PAGMO package [16]. The population consists of 50 individuals and is evolved for 400 generations. Selection is done through a tournament selection strategy, mating through exponential crossover and mutation using a polynomial mutation 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. The evolved parameters will be further discussed in Section IV.

For each agent, its cost is defined as its average distance to source added by a penalty (+ 1.0) in the event of a collision. Even without a penalty the agents will learn to avoid obstacles to some capacity, though a penalty will force the agent to be more careful. Other quality metrics like 'time to source' were also considered, but we found average distance to work best and to be most objective. Average distance to source will lead to finding the most direct paths to the source, and then stay close to the source. This could be useful in emergency

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

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 III-B FOR THE MEANING OF THE VARIABLES.

response, when a human comes in and immediately sees multiple drones hover close to a peak in gas concentration.

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' [31] on performance in simulation. When using doping, the probability of environment number i to be selected in each draw is described by Equation 7.

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

$D(i)$ is the 'difficulty' of environment i, computed based on previous experience with environment i. 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.

Probability $P(i)$ is defined as the difficulty of environment i, divided by the sum of all difficulties: $\sum_{k=0}^{m} D(k)$. This means 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.

IV. RESULTS

In this section, we evaluate simulation and flight results of our swarm. We evaluate training results (Section IV-A), evaluate the models in simulation (Section IV-B), and flight tests (Section IV-C).

A. Training in Simulation

To evolve our agents, we use AutoGDM to randomly generate 100 environments of 10×10 m in size, the size of our experimental arena. We use 3 agents, with $V_{desired} = 0.5\,\mathrm{m/s}$. Not only the environment configurations, but also the boundary conditions are randomized, arriving at varying gas dispersion conditions. Figure 5 shows two randomly generated environments. The left environment contains few obstructions and shows a stable and diffusion-dominated gas source, whereas the right environment is less stable and contains more local optima. By evolving Sniffy Bug on a heterogeneous set of environments, we arrive at a set of parameters that works in different gas and environment configurations.

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 s, to reduce the computational cost. The evaluations are done in parallel on an Intel(R) Core(TM) i7-7700 CPU, and all headings and starting positions are randomized after each generation. Agents are spawned in some part of the area so that some path exists towards the gas source, and they do not get spawned inside a closed room.

We assess training results for training using doping. Figure 6 shows the progress of the cost function during training. Table I shows the parameters resulting from evolution in comparison with the manually designed parameters. r_{range} is set to 10 m, creating random waypoints in a box of 10 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 $\mathbf{s}_j$, to move away from its personal best-seen position $\mathbf{p}_{i,j}$, and move 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 any other agent is within 0.782 m. This is substantially lower than the manual number of 1.5 m, which can be explained by a lower cost function when agents stay close to each other when they found the source. It also makes it easier to pass through narrow passages, but could increase the risk of collision when agents are close to each other for extended periods of time.

d_{wp} should also be highlighted, as 2.69 m is much higher than our manual choice. Instead of using the timeout to generate new waypoints, the evolved version of Sniffy Bug uses PSO to determine the desired direction to follow, until it has travelled far enough in the desired direction and generates a new waypoint. It looks like it has practically disabled the timer (t_{wp} = 51.979) in favor of following the desired direction, instead of waypoints that can be reached.

From an obstacle avoidance point of view, we see the manual parameters are more conservative as compared to the evolved counterparts. Being less conservative allows the agents to get closer to the source quicker, at the cost of an increased risk of collision. This is an interesting trade-off: if we deploy larger swarms in the real world (i.e., $>$ 10 agents), how safe do they need to be? Or is the extra risk worth more efficient localization? The trade-off can be shifted by introducing a higher penalty in the cost function.

After training, the probability for each environment in each draw can be evaluated as a measure of difficulty. Figure 16 shows a histogram of all 100 probabilities along with some environments on the spectrum. Generally, more cluttered environments with more local minima result in a higher $D(i)$ and $P(i)$, thus are perceived as more difficult.

B. Evaluation in Simulation

Next, we evaluate the models separately in simulation on test environments. Figures 7 and 8 show runs in simulation of the manual and evolved version with doping respectively, starting for the same positions with the same headings and in the same conditions.

Our hypothesis stated in Section IV-A is fortified by simulation experiments: the evolved policy shows more direct paths by, among other things, being less conservative. Both algorithms find the source, though the evolved counterpart

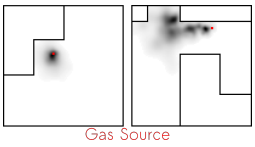


Fig. 5. Gas dispersed in two randomly generated environments.

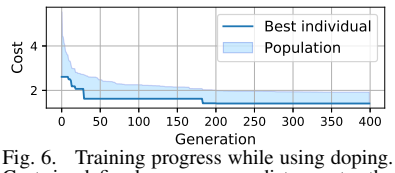


Fig. 6. Training progress while using doping. Cost is defined as average distance to the source, added by a small collision penalty.

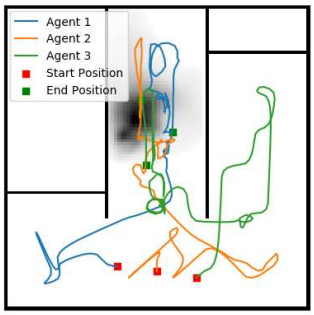


Fig. 7. Sniffy Bug using manual parameters, successfully locating the source.

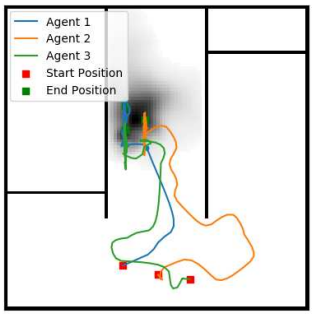


Fig. 8. Sniffy Bug using parameters that are evolved using doping, successfully finding the source using more direct paths.

finds it quicker and stays closer to the source. Evolved Sniffy Bug allows the agents to get closer to each other, resulting in a final position very close to the source.

To further analyse performance, and assess the effect of doping, we evaluate all three controllers in all 100 environments, and record different quality 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 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 inferior to the manual parameters. This is caused by the hard instance problem [31]: the parameters are optimized to perform best on average rather than to perform well in all cases.

	Success Rate	Avg Distance to Source [m]	Avg time to source [s]
Manual Parameters	89 %	3.29	51.1
Evolved without Doping	85 %	2.90	47.1
Evolved with Doping	**93 %**	**2.73**	**39.2**

TABLE II
SNIFFY BUG EVALUATED ON 100 RANDOMLY GENERATED ENVIRONMENTS.

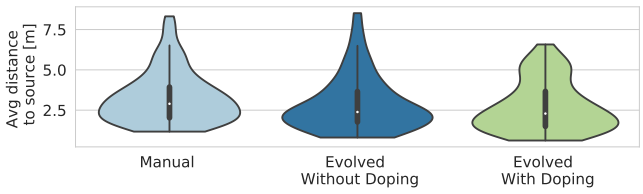


Fig. 9. Three sets of parameters evaluated in simulation: 1) manually set parameters, 2) parameters evolved without doping, and 3) parameters evolved using doping [31] to address the hard instance problem.

On the other hand, the parameters that are evolved with doping outperform the other two 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 9. The doping controller does not only result in a lower average distance to source, it also shows a much smaller spread. Doping managed to eliminate the long tail that is present in the set of evolved parameters, as shown in Figure 9.

As we test on a finite set of environments, and the assumptions of normality may not hold, we use the empirical bootstrap [12] to compare the means of the average distance to source. Using 100,000 bootstrap iterations, we find that only when comparing the manual parameters with evolved parameters with doping, the null hypothesis can be rejected, with $P = 0.0062$. From these simulation results, it looks like doping is required to arrive at parameters that are significantly better than manual parameters.

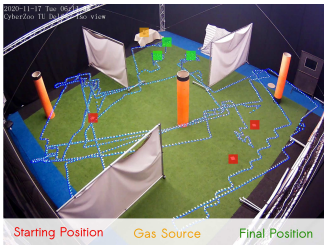
C. Flight Tests

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 10. We place a small can of alcohol with a 5V computer fan within the room, which the swarm will attempt to locate. We compare manual parameters against the parameters that are evolved using doping, by comparing their recorded gas readings. Each set 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 $120\,\mathrm{s}$. Figure 11 corresponds to the run depicted in Figure 10a, Figures 13-15 show example runs of Environments 2-4.

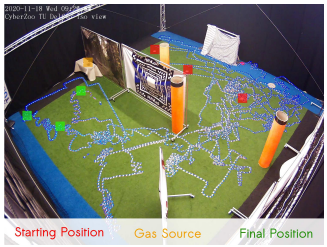
Figure 12 shows the maximum recorded gas concentration by the swarm, for each time step for each run. Especially for environment 1, it clearly shows the more efficient source seeking behavior of our evolved controller. Table III shows the average and maximum observed concentrations by the swarm, averaged 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 $\pm 2.0\,\mathrm{m}$ 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), likely thanks to it spending less time around dangerous obstacles, thanks to more efficient GSL.

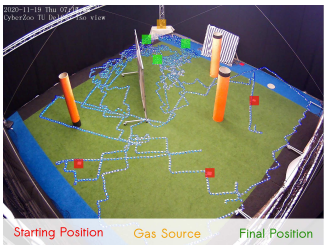
On the other hand, the evolved parameters recorded 2 crashes, both in environment 4, when the agents were really close to each other and the source for extended periods of time. The manual parameters result in more distance between agents, making it more robust against downwash and momentarily poor relative 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, $0.8\,\mathrm{m}$ away from each other instead of $0.5\,\mathrm{m}$.



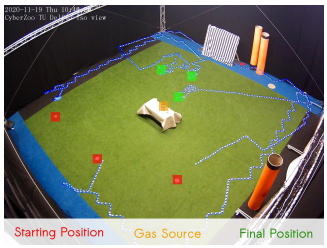
(a) Environment 1: the agents need to explore the 'room' in the corner to smell the gas.



(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 empty environment.

Fig. 10. Sniffy Bug evaluated on four distinct environments, 10 × 10 m in size, seeking a real alcohol source.

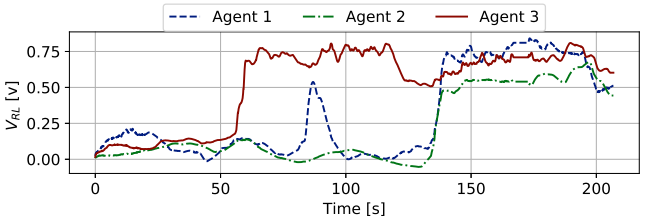
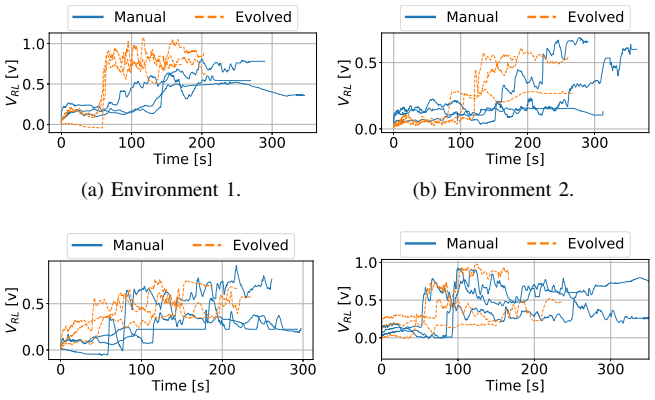


Fig. 11. Evolved Sniffy Bug seeking an alcohol gas source in environment 1. After agent 3 finds the source, all three agents quickly find their way to higher concentrations.

The results show that AutoGDM can be used to evolve a controller that works in the real world in challenging conditions. While GADEN [25] and OpenFOAM [17] were validated, our approach of randomization of source position and boundary condition had not yet been validated. Additionally, the results show that the simple particle motion model and sensor model (i.e., feeding concentrations straight from AutoGDM) are sufficient to evolve a controller that works in the real world. We demonstrate that fully automated randomization of the environment, source position and boundary conditions, results in a high-performance solution that

	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
MANUAL PARAMETERS AND PARAMETERS EVOLVED WITH DOPING COMPARED IN 24 FLIGHT TESTS. SHOWN ARE THE AVERAGE AND MAXIMUM SMELLED CONCENTRATION BY THE SWARM, AVERAGED FOR THE THREE RUNS FOR EACH ENVIRONMENT.



(a) Environment 1. (b) Environment 2.

(c) Environment 3. (d) Environment 4.

Fig. 12. Maximum recorded gas reading by the swarm, for each time step for each run.

outperforms manual parameters in the real world. Hereby, we demonstrate zero-shot sim2real transfer for GSL, contributing the first controller for GSL that was evolved in a simulator and tested in challenging real-world environments.

V. Conclusion and Discussion

We show that a novel and minimal bug algorithm, Sniffy Bug, resulted in the first fully autonomous swarm of gas-seeking nano quadcopters. We evolve the parameters of this algorithm, which covers both obstacle avoidance and navigation. It outperforms a human-designed controller in all metrics in simulation and corresponding robot experiments. With this, we contribute the first transfer of a learning-based GSL solution from simulation to a swarm of nano quadcopters, demonstrating zero-shot sim2real transfer.

Furthermore, we contribute the first fully automated GDM pipeline, AutoGDM, that does not only allow for learning in simulation, but is also useful in benchmarks. Most previous works use disparate and simplified models, making comparison virtually impossible. Not only will AutoGDM help arrive at robust algorithms that work in challenging environments, it also serves as a platform to better understand the performance of existing contributions. A set of benchmark environments will soon be available in a public repository[2].

In future work, we hope to see our methodology tested in larger and more complex real-world environments, with multiple rooms and complex obstacles. PSO was designed to work in large optimization problems with many local optima, making it likely that our methodology extends to more complex configurations. Even more, AutoGDM may be used to enable reinforcement learning for GSL, by training a policy in simulation with AutoGDM that can be deployed in the real world. In this study, we have opted for a FSM, that effectively reduces the sim2real gap by tracking waypoints with PSO. An end-to-end learning approach with appropriate input may outperform our method.

Nonetheless, we are convinced that this work is an important step in the right direction. Insects use simple strategies to solve complex problems and survive in nature. We show how a simple algorithm and application-specific system design can be used to solve the complex multi-agent GSL problem.

[2]https://github.com/tudelft/sniffy_bug

APPENDIX A
ADDITIONAL ALGORITHM

Algorithm 1: Sniffy Bug Wall Following

```
// determine desired laser and search direction (see Figure 3)
desired_laser = determine_desired_laser(WP_Heading);
search_clockwise = determine_direction(desired_laser,WP_Heading);

// initialize wall following variable
max_reached_laser = desired_laser;

// initialize for avoided_obs detection
exited_green_zone = false ;
init_dist = get_distance(current_pos,goal_wp) ;
wp_line = get_line(current_pos,goal_wp) ;
avoided_obs = false ;

while Wall_Following do
    get_laser_readings(lasers);

    if get_distance_to_line(wp_line,current_pos) > d_line then
        exited_green_zone = true ;
    else
        if exited_green_zone == true and
          get_distance(current_pos,goal_wp) < init_dist then
            avoided_obs = true ;
        end
    end

    if search_clockwise then
        for (i=desired_laser; i>(desired_laser-4);
          i--) do
            if i < max_reached_laser then
                max_reached_laser = i ;
            end
            if i ≤ (max_reached_laser + 1) then
                // bounding local_laser between 0 and 3
                local_laser = cap_laser(i);
                if lasers[local_laser] > d_laser then
                    break;
                end
            end
        end
        if lasers[local_laser] < d_laser then
            search_clockwise = false ;
            max_reached_laser = desired_laser;
        end
    else
        for (i=desired_laser; i<(desired_laser+4);
          i++) do
            if i > max_reached_laser then
                max_reached_laser = i ;
            end
            if i ≥ (max_reached_laser - 1) then
                // bounding local_laser between 0 and 3
                local_laser = cap_laser(i);
                if lasers[local_laser] > d_laser then
                    break;
                end
            end
        end
        if lasers[local_laser] < d_laser then
            search_clockwise = true ;
            max_reached_laser = desired_laser;
        end
    end
    if new_wp_received or agent_close or avoided_obs then
        Wall_Following = false;
    else
        fly(local_laser);
    end
end
```

APPENDIX B
ADDITIONAL FIGURES

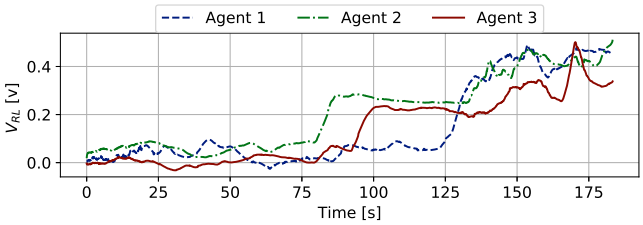


Fig. 13. Evolved Sniffy Bug seeking an alcohol gas source in environment 2. After agent 2 finds the source, all three agents quickly find their way to higher concentrations.

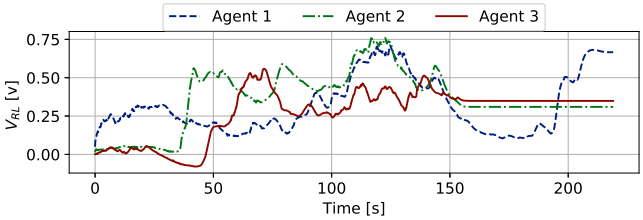


Fig. 14. Evolved Sniffy Bug seeking an alcohol gas source in environment 3. After agent 1 finds the source, all three agents quickly find their way to higher concentrations.

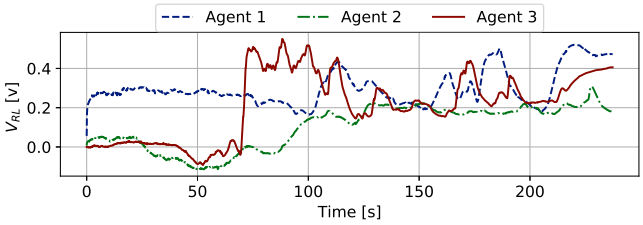


Fig. 15. Evolved Sniffy Bug seeking an alcohol gas source in environment 4. After agent 3 finds the source, all three agents quickly find their way to higher concentrations.

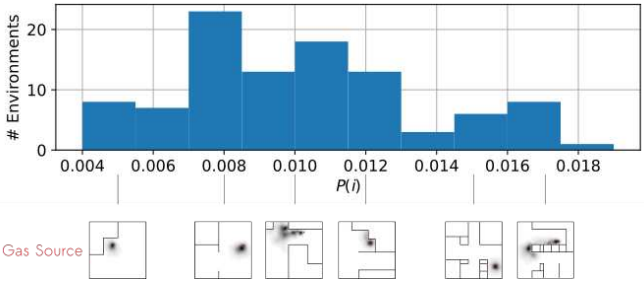


Fig. 16. Doping probabilities, showing the distribution of computed probabilities for each environment at the end of evolution (generation 400). The environments below the x-axis show that harder environments contain more obstacles and local optima.

References

[1] Julius Adler. The sensing of chemicals by bacteria. *Scientific American*, 234(4):40–47, 1976. ISSN 00368733, 19467087. URL http://www.jstor.org/stable/24950326.

[2] Melanie Joyce Anderson, Joseph Garret Sullivan, Timothy Horiuchi, Sawyer Buckminster Fuller, and Thomas L Daniel. A bio-hybrid odor-guided autonomous palm-sized air vehicle. *Bioinspiration Biomimetics*, 2020. URL http://iopscience.iop.org/article/10.1088/1748-3190/abbd81.

[3] Martin Asenov, Marius Rutkauskas, Derryck Reid, Kartic Subr, and Subramanian Ramamoorthy. Active localization of gas leaks using fluid simulation. *IEEE Robotics and Automation Letters*, 4(2):1776–1783, 2019. ISSN 23773766. doi: 10.1109/LRA.2019.2895820.

[4] Randall D. Beer and John C. Gallagher. *Evolving Dynamical Neural Networks for Adaptive Behavior*, volume 1. 1992. ISBN 1059712392001. doi: 10.1177/105971239200100105.

[5] J. R. Bourne, E. R. Pardyjak, and K. K. Leang. Coordinated bayesian-based bioinspired plume source term estimation and source seeking for mobile robots. *IEEE Transactions on Robotics*, 35(4):967–986, 2019.

[6] Javier Burgués, Victor Hernández, Achim J. Lilienthal, and Santiago Marco. Smelling nano aerial vehicle for gas source localization and mapping. *Sensors (Switzerland)*, 19(3), 2019. ISSN 14248220. doi: 10.3390/s19030478.

[7] Mohammadreza Chamanbaz, David Mateo, Brandon M. Zoss, Grgur Tokí, Erik Wilhelm, Roland Bouffanais, and Dick K.P. Yue. Swarm-enabling technology for multi-robot systems. *Frontiers Robotics AI*, 4(APR):1–12, 2017. ISSN 22969144. doi: 10.3389/frobt.2017.00012.

[8] Xinxing Chen and Jian Huang. Combining particle filter algorithm with bio-inspired anemotaxis behavior: A smoke plume tracking method and its robotic experiment validation. *Measurement*, 154:107482, 2020. ISSN 0263-2241. doi: https://doi.org/10.1016/j.measurement.2020.107482. URL http://www.sciencedirect.com/science/article/pii/S0263224120300191.

[9] John Clegg. Signals in the animal world, by d. burkhardt, w. schleidt and h. altner. allen and unwin, 63s. *Oryx*, 9 (5):372–373, 1968. doi: 10.1017/S0030605300007122.

[10] Mario Coppola, Kimberly N. McGuire, Christophe De Wagter, and Guido C. H. E. de Croon. A survey on swarming with micro air vehicles: Fundamental challenges and constraints. *Frontiers in Robotics and AI*, 7:18, 2020. ISSN 2296-9144. doi: 10.3389/frobt.2020.00018. URL https://www.frontiersin.org/article/10.3389/frobt.2020.00018.

[11] G. C.H.E. de Croon, L. M. O'Connor, C. Nicol, and D. Izzo. Evolutionary robotics approach to odor source localization. *Neurocomputing*, 121(December):481–497, 2013. ISSN 09252312. doi: 10.1016/j.neucom.2013.05.028.

[12] B. Efron. Bootstrap methods: Another look at the jackknife. *Ann. Statist.*, 7(1):1–26, 01 1979. doi: 10.1214/aos/1176344552. URL https://doi.org/10.1214/aos/1176344552.

[13] J. E. Fackrell and A. G. Robins. Concentration fluctuations and fluxes in plumes from point sources in a turbulent boundary layer. *Journal of Fluid Mechanics*, 117:1–26, 1982. doi: 10.1017/S0022112082001499.

[14] Eduardo J. Izquierdo and Thomas Buhrmann. Analysis of a dynamical recurrent neural network evolved for two qualitatively different tasks: Walking and chemotaxis. *Artificial Life XI: Proceedings of the 11th International Conference on the Simulation and Synthesis of Living Systems, ALIFE 2008*, pages 257–264, 2008.

[15] Eduardo J. Izquierdo and Shawn R. Lockery. Evolution and analysis of minimal neural circuits for klinotaxis in Caenorhabditis elegans. *Journal of Neuroscience*, 30(39):12908–12917, 2010. ISSN 15292401. doi: 10.1523/JNEUROSCI.2606-10.2010.

[16] Dario Izzo, Marek Ruciński, and Francesco Biscani. *The Generalized Island Model*, volume 415, pages 151–169. 01 2012. ISBN 978-3-642-28788-6. doi: 10.1007/978-3-642-28789-3_7.

[17] Hrvoje Jasak. Openfoam: Open source cfd in research and industry. *International Journal of Naval Architecture and Ocean Engineering*, 1(2):89 – 94, 2009. ISSN 2092-6782. doi: https://doi.org/10.2478/IJNAOE-2013-0011. URL http://www.sciencedirect.com/science/article/pii/S2092678216303879.

[18] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

[19] Yoshihiko Kuwana, Isao Shimoyama, Yushi Sayama, and Hirofumi Miura. Synthesis of pheromone-oriented emergent behavior of a silkworm moth. *IEEE International Conference on Intelligent Robots and Systems*, 3:1722–1729, 1996. doi: 10.1109/iros.1996.569043.

[20] Yoshihiko Kuwana, Sumito Nagasawa, Isao Shimoyama, and Ryohei Kanzaki. Synthesis of the pheromone-oriented behaviour of silkworm moths by a mobile robot with moth antennae as pheromone sensors. this paper was presented at the fifth world congress on biosensors, berlin, germany, 3–5 june 1998.1. *Biosensors and Bioelectronics*, 14(2):195 – 202, 1999. ISSN 0956-5663. doi: https://doi.org/10.1016/S0956-5663(98)00106-7. URL http://www.sciencedirect.com/science/article/pii/S0956566398001067.

[21] Shushuai Li, Mario Coppola, Christophe De Wagter, and Guido C. H. E. de Croon. An autonomous swarm of micro flying robots with range-based relative localization, 2020.

[22] Thomas Lochmatter and Alcherio Martinoli. *Understanding the Potential Impact of Multiple Robots in Odor Source Localization*, pages 239–250. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-00644-9. doi: 10.1007/978-3-642-00644-9_21. URL https://doi.org/10.1007/978-3-642-00644-9_21.

[23] 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*, 4(35), 2019. doi: 10.1126/scirobotics.aaw9710. URL https://robotics.sciencemag.org/content/4/35/eaaw9710.

[24] K.N. McGuire, G.C.H.E. de Croon, and K. Tuyls. A comparative study of bug algorithms for robot navigation. *Robotics and Autonomous Systems*, 121:103261, 2019. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2019.103261. URL http://www.sciencedirect.com/science/article/pii/S0921889018306687.

[25] Javier Monroy, Victor Hernandez-Bennetts, Han Fan, Achim Lilienthal, and Javier Gonzalez-Jimenez. Gaden: A 3d gas dispersion simulator for mobile robot olfaction in realistic environments. *MDPI Sensors*, 17(7:1479):1–16, 2017. ISSN 1424-8220. doi: 10.3390/s17071479. URL http://mapir.isa.uma.es/mapirwebsite/index.php/mapir-downloads/papers/259.

[26] Eduardo Martin Moraud and Dominique Martinez. Effec-

tiveness and robustness of robot infotaxis for searching in dilute conditions. *Frontiers in Neurorobotics*, 4(MAR): 1–8, 2010. ISSN 16625218. doi: 10.3389/fnbot.2010.00001.

[27] 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*, 33(5):1255–1262, 2017. doi: 10.1109/TRO.2017.2705103.

[28] M. Rossi and D. Brunelli. Gas sensing on unmanned vehicles: Challenges and opportunities. In *2017 New Generation of CAS (NGCAS)*, pages 117–120, 2017. doi: 10.1109/NGCAS.2017.58.

[29] Debajit Saha, Darshit Mehta, Ege Atlan, Rishabh Chandak, Mike Traner, Ray Lo, Prashant Gupta, Srikanth Singamaneni, Shantanu Chakrabartty, and Barani Raman. Explosive sensing with insect-based biorobots. *bioRxiv*, 2020. doi: 10.1101/2020.02.10.940866. URL https://www.biorxiv.org/content/early/2020/02/11/2020.02.10.940866.

[30] Cheng Song, Yuyao He, Branko Ristic, and Xiaokang Lei. Collaborative infotaxis: Searching for a signal-emitting source based on particle filter and Gaussian fitting. *Robotics and Autonomous Systems*, 125: 103414, 2020. ISSN 09218890. doi: 10.1016/j.robot.2019.103414. URL https://doi.org/10.1016/j.robot.2019.103414.

[31] Pieter Spronck, Ida Sprinkhuizen-Kuyper, and Eric Postma. Deca: The doping-driven evolutionary control algorithm. *Applied Artificial Intelligence*, 22:169–197, 03 2008. doi: 10.1080/08839510701527309.

[32] Jake A. Steiner, Joseph R. Bourne, Xiang He, Donald M. Cropek, and Kam K. Leang. Chemical-source localization using a swarm of decentralized unmanned aerial vehicles for urban/suburban environments. *ASME 2019 Dynamic Systems and Control Conference, DSCC 2019*, 3(February 2020), 2019. doi: 10.1115/DSCC2019-9099.

[33] Massimo Vergassola, Emmanuel Villermaux, and Boris I. Shraiman. 'Infotaxis' as a strategy for searching without gradients. *Nature*, 445(7126):406–409, 2007. ISSN 14764687. doi: 10.1038/nature05464.

[34] Nicole Voges, Antoine Chaffiol, Philippe Lucas, and Dominique Martinez. Reactive searching and infotaxis in odor source localization. *PLOS Computational Biology*, 10(10):1–13, 10 2014. doi: 10.1371/journal.pcbi.1003861. URL https://doi.org/10.1371/journal.pcbi.1003861.

[35] Jian Long Wei, Qing Hao Meng, Ci Yan, Ming Zeng, and Wei Li. Multi-Robot gas-source localization based on reinforcement learning. *2012 IEEE International Conference on Robotics and Biomimetics, ROBIO 2012 - Conference Digest*, pages 1440–1445, 2012. doi: 10.1109/ROBIO.2012.6491171.

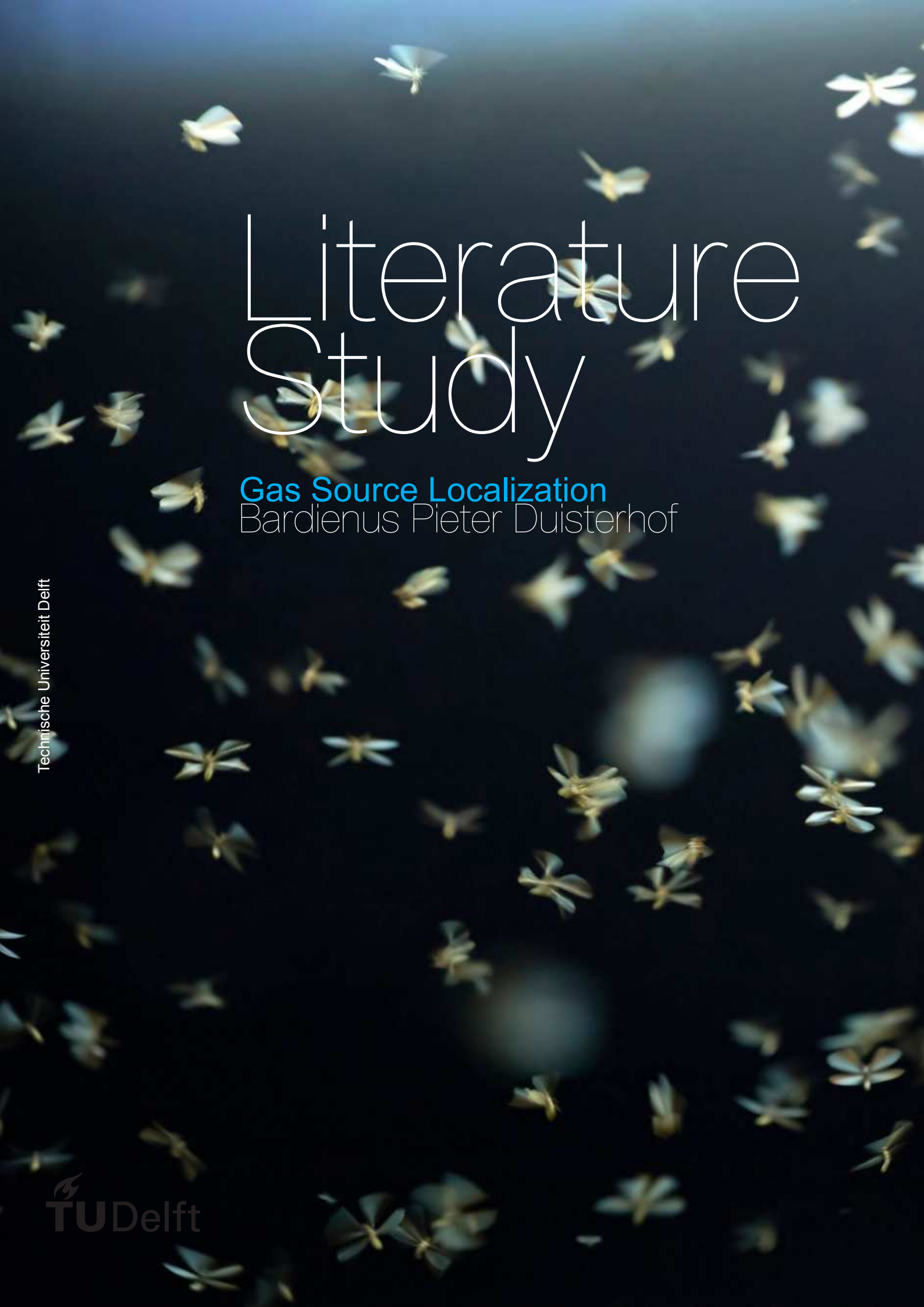
[36] Thomas Wiedemann, Dmitriy Shutin, and Achim J. Lilienthal. Model-based gas source localization strategy for a cooperative multi-robot system - A probabilistic approach and experimental validation incorporating physical knowledge and model uncertainties. *Robotics Auton. Syst.*, 118:66–79, 2019. doi: 10.1016/j.robot.2019.03.014. URL https://doi.org/10.1016/j.robot.2019.03.014.

[37] Frieder Wittmann, Olivier Lambercy, and Roger Gassert. Magnetometer-based drift correction during rest inimu arm motion tracking. *Sensors (Basel, Switzerland)*, 19(6), March 2019. ISSN 1424-8220. doi: 10.3390/s19061312. URL https://europepmc.org/articles/PMC6471153.

[38] Xin xing Chen and Jian Huang. Odor source localization algorithms on mobile robots: A review and future outlook. *Robotics and Autonomous Systems*, 112 (December):123–136, 2019. ISSN 09218890. doi: 10.1016/j.robot.2018.11.014.

[39] Hao Xu, Luqi Wang, Yichen Zhang, Kejie Qiu, and Shaojie Shen. Decentralized visual-inertial-uwb fusion for relative state estimation of aerial swarm. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020. doi: 10.1109/icra40945.2020.9196944. URL http://dx.doi.org/10.1109/ICRA40945.2020.9196944.

Literature Study



Literature Study

Gas Source Localization

Bardienus Pieter Duisterhof

Technische Universiteit Delft

TU Delft

Literature Study

Gas Source Localization

by

Bardienus Pieter Duisterhof

October 5 2020

TU Delft

Contents

1 Introduction

Gas source localization (GSL) is a highly sought-after feature in the field of robotics, as it has the potential to defer risk away from humans in presence of gas leaks. GSL is defined as localizing a gas source by one or more mobile robots, generally by gas sensors, as most gases are invisible to cameras. The ideal gas-seeking robot is agile, small and cheap, for effective deployment. A nano drone might be the perfect candidate, but its resource constraints have so far resulted in limited success. This study reviews literature on the topic of gas seeking, to identify the key challenges to enable a swarm of gas-seeking nano drones. The existing literature can be roughly divided into four categories, being: 1) single-agent algorithms, 2) multi-agent algorithms, 3) reinforcement learning and 4) gas modelling.

Single agent algorithms are classified as either bio-inspired, probabilistic or gas distribution mapping (GDM). Bio-inspired algorithms draw inspiration from organisms such as silkworm moths [60], E. coli bacteria [1] or dung beetles [16]. These algorithms can be divided into three stages: 1) plume acquisition, 2) reactive plume tracking, and 3) source declaration. Only a small subset of algorithms solves all three stages. The plume acquisition stage can be completed by a random walker [3], drawing actions from a Poisson distribution. Other options include scanning patterns with fixed orientation with respect to the wind vector, such as a linear search [16].

For reactive plume tracking, algorithms heavily relying on chemical gradient are generally less stable but more efficient in stable diffusion-like conditions [140]. The E. coli [1] inspired algorithm is an example of a strategy with heavy reliance on gradient, as it determines its next heading based on chemical gradient only. The more successful moth-inspired algorithms [60] do not just use chemical measurements, but also wind-vector information. They are inspired from the strategy a male moth deploys to mate, as the female disperses odours. Comparative studies [74, 75] have shown that surge-cast [60] and surge-spiral [38, 75] are the most successful bio-inspired algorithms in real robot tests. The robot flies straight upwind (surge) until it loses the plume, so it starts oscillating perpendicular to the wind flow (casting) until it finds the plume again. In surge-spiral [38, 75], casting is replaced by an archimedean spiral. Even though these algorithms perform relatively well, they are not designed for complex multi-room environments and are not easily extended to multi-agent setups. The final stage, source declaration, has limited contributions. The most simple strategy is to set a gas concentration threshold, but more sophisticated algorithm-specific methods exist.

Probabilistic-based strategies were designed to be more robust in turbulent conditions, though it remains unclear if that is true in real-world experiments [131]. Infotaxis [130] constructs a 'belief map', keeping track of spatial probability of source presence, based on a history of chemical and wind readings. Every next step is taken to decrease uncertainty in source position, the agent is seeking information. After it was introduced in 2007, the community has quickly adopted, introducing more probabilistic methods and implementations on robots. Source term estimation (STE) [27] is a popular subset of probabilistic algorithms, fitting measurements to a pre-defined model usually using some form of Bayesian inference. The primary challenge for implementation on a nano drone is the high computational cost and memory requirement. Additionally, probabilistic algorithms use simplified plume models, giving poor predictions in complex environments. Probabilistic methods have not yet considered obstacles in their plume models, they simply lack information.

Gas distribution mapping (GDM) [67] is another solution, focusing on the creation of a map of the

entire area, instead of optimizing efficiency in localization. Advantages include the guaranteed localization of the source, while the efficiency takes a big hit.

Multi-agent approaches are useful when deployed in emergency response, as robustness and efficiency are key to mission success. Increasing the number of agents improves robustness of the entire system, while locating the source faster. Particle swarm optimization (PSO) [54] is frequently used for multi-agent GSL, many works contributed modifications to improve performance. Glowworm Swarm Optimization (GSO) [58] was used too, which, contrary to PSO, is capable of identifying multiple sources. While PSO has the tendency to converge to local optima, GSO can converge to different local optima in groups. Bio-inspired algorithms can be deployed with multiple robots, especially the moth-inspired spiral-surge algorithm has been studied in a multi-agent context, though its performance was limited [76]. Formation-based swarming [15] is another useful strategy, exerting repulsive and attractive forces in between agents. An attractive force applied towards the source can simply be chemical gradient, or something more complex like mass flux, also called fluxotaxis [142].

When deploying probabilistic algorithms in a multi-agent setting, the exchange of data within strict communication resource constraints can be challenging. Early works assume all-to-all communication, communicating all measurements, while later contributions fit a Gaussian distribution to the source likelihood map and communicate the mean and variance of that distribution [116].

The last category of GSL algorithms considered in this work are reinforcement learning (RL) algorithms. RL algorithms figure out a way to optimize their policy through a user-defined reward. Gradient-free algorithms (e.g., neuroevolution), deploy evolutionary algorithms to optimize the weights of a neural network based on simulation results. Innovations in gradient-free reinforcement learning include indirect encoding [122], evolving the topography [121] and meta learning [99]. Gradient-based RL algorithms optimize the policy directly based on state and actions pairs, and is more popular for high-dimensional inputs.

In a multi-agent setup RL has been successful too, as agents are capable of learning to communicate, deciding when and what to communicate [53]. Some of this theory has been applied to GSL, first in [9, 59], later in [19, 46, 47, 133]. The results in simulation look very promising, but no exhaustive real-world testing of a RL algorithm for GSL has been performed up to this point.

We have now introduced all algorithm categories considered, Table 1.1 shows a comparison between all the considered algorithm groups in this survey, evaluated for deployment onboard a nano drone.

	Bio-Inspired	**Probabilistic**	**MA-Specific**	**GDM**	**(RL)**
Computationally cheap	V	X	V	V	V
Efficient in:					
Large open space	V	V	V	X	V
Turbulent flow	V	V	V	X	V
Complex environment	X	X	X	X	V
Extends well to MA	X	V	V	V	V

Table 1.1: Properties of different solutions to GSL, bio-inspired, probabilistic, MA-specific, gas distribution mapping (GDM) and a potential reinforcement learning strategy (speculation). Bio-inspired algorithms are the algorithms directly inspired from gas-seeking creatures. MA-specific algorithms are the non-probabilistic and non-RL algorithms only used for a multi-agent set up, like PSO and BESA.

Comparison of these algorithms is hard, as most works provide their own gas simulations, making it an unfair comparison. Gas models deployed in previous GSL contributions can be divided in Gaussian Plume Models (GPM) [141] and advection and diffusion based models [96]. GPM's model concentration as a Gaussian curve, increasing its standard deviation further away from the source. In advection and diffusion based modelling, particles are released in a flow field, traced (advection) and expanded over time (diffusion). Computational fluid dynamics (CFD) is the most accurate method for generation of the flow field in complex environments. In previous works custom solutions were developed to this problem, though especially for the success of RL-based GSL, *there is a need for an end-to-end simulation platform to efficiently generate and model gas dispersion in complex environments.*

In this review, we point out four main research gaps:

- No real robot experiments exist with reinforcement learning algorithms for GSL, except for one very small-scale experiment in [59].

- The vast majority of all contributions test their strategies in a small-scale environment without obstacles, only two very recent [28, 125] large-scale experiments with obstacles exist. These contributions could locate a source in larger areas with obstacles, though their efficiency is unknown.
- Comparison between models is almost impossible at this point, as each work tests the algorithms in a different environment under different conditions. There is a need for a software platform for end-to-end environment generation and dispersion modelling, which would not only allow for learning in simulation, but would also provide a useful benchmark.
- Key factors in the success of a GSL robot are cost and agility, making a nano drone the perfect candidate. So far, only [14] has contributed a GSL implementation on the nano scale. [14] deploys the relatively unknown signal processing strategy proposed in [107], and demonstrates successful GSL on a nano drone while using an environment map and external positioning. GSL on a nano drone introduces its own unique challenges, hence more research is required to find the most robust, efficient and fully autonomous GSL strategy for nano drones.

The remainder of this review is structured as follows: Chapter 2 introduces single agent algorithms, Chapter 3 multi-agent algorithm and Chapter 4 reinforcement learning algorithms. Chapter 5 provides an overview on relevant gas modelling methods, and Chapter 6 concludes the review.

Figure 1.1: A small simulated environment from [135], with hardware in the loop.

Figure 1.2: Large complex environment with obstacles from [28].

2

Single-Agent Gas Source Localization

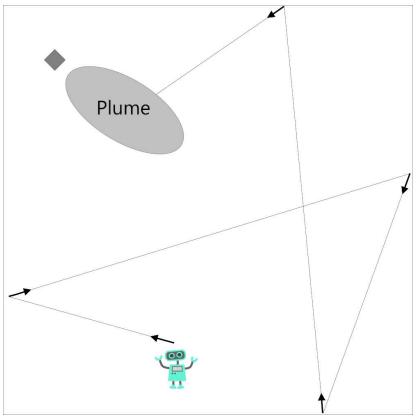
This chapter provides an overview of all single-agent odor source localization algorithms. We subdivide into two separate classes, being bio-inspired (Section 2.1) and probabilistic/map-based (Section 2.2). Finally, we review source declaration methods 2.3.

2.1. Bio-Inspired

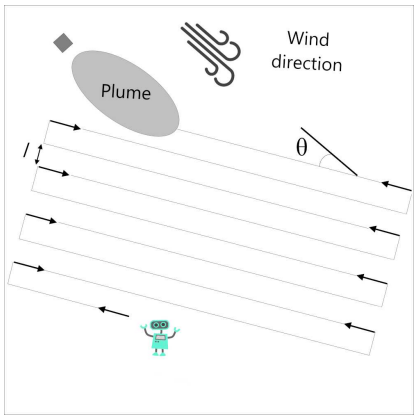
Early work in the field of gas source localization (GSL) was primarily bio-inspired [100], modeled after organisms such as the silkworm moths [60] or E. coli bacteria [1]. We first compare strategies capable of locating the plume, when no chemical readings are present yet. Once the plume has been located, chemotaxis and anemotaxis strategies are compared, using only chemical or chemical and wind information respectively.

2.1.1. Plume Acquisition

Before any chemical signal is detected, the robot will need to find the edge of the plume. Three bio-inspired methods are known: 1) passive monitoring [52], 2) random walk [26] and 3) linear search [16]. In passive monitoring, the agent simply waits before a moving source is detected, when it starts following the target. This strategy was observed in male moths, waiting for an odor of pheromone released by the female moth [52].



(a) Random walk plume acquisition method. The agent keeps traveling forward until it finds an obstacle and rotates a random angle.



(b) Linear search plume acquisition at an angle θ w.r.t. the wind vector, in a scanning pattern separated by distance l.

Figure 2.1: Random walk and linear search plume acquisition strategies demonstrations.

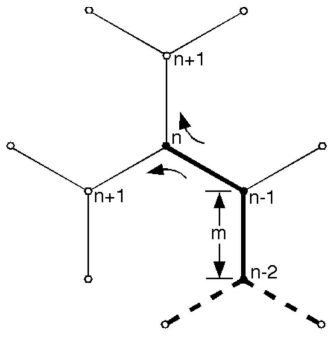


Figure 2.2: Hex path algorithm, deciding to move clockwise or anti-clockwise at n based on readings at $n-1$ and $n-2$.

The most straight-forward active searching strategy is a random walk (Figure 2.1a), which moves forward until it encounters an obstacle, and then rotates for a random angle. The resulting behavior

is comparable to Brownian motion [23]. In the E. coli-inspired algorithm [1], every step has a random length and heading change, according to a Poisson distribution. In some cases the distribution is biased by chemical gradient in reactive plume tracking [3].

Finally, in a linear search (Figure 2.1b) the agent deploys a scanning pattern at an angle θ w.r.t. the wind vector and separated by a distance l. This strategy has been observed in dung beetles, and is theoretically most effective when $\theta = 90°$. Generally, from [105], a linear search results in a shorter mean path length as compared to a random walker. This is interesting from an evolutionary point of view, as the dung beetle seems to have evolved to use its wind information to be more effective as compared to the E. coli bacteria.

2.1.2. Chemotaxis

Chemotaxis is defined as the movement of a cell or organism towards concentration of a substance. It was first discovered in a plant's (bracken fern) spermatozoa [98]. Not only has it proven to be essential in the beginning of life, it also a key mechanism in injury recovery and immune responses [136]. While the exact definition of chemotaxis in the field of robotics is ambiguous, it is usually referred to as algorithms using a chemical gradient and/or reading to locate an odor source.

In this section, we refer to chemotactic methods when they rely solely on information from a chemical sensor. The three single-agent chemotaxis algorithms reviewed are: 1) Braitenberg Vehicles, 2) Hex-Path algorithm and 3) E. coli inspired methods.

Braitenberg Vehicles

Braitenberg introduced the Braitenberg vehicles in 1986 [13], autonomous agents with a direct coupling between some sensor and actuator (wheels). He demonstrated phototaxis by a simple autonomous agent, that was surprisingly successful, given its simple strategy. Different Braitenberg vehicles were presented in [13], establishing different couplings between two bilateral sensors. For chemotaxis, two chemical sensors can be connected through inhibitory cross-coupling or excitatory "same side" connections. The robot now simply turns towards the side with highest sensor reading. [68] compared different Braitenberg vehicles for GSL, and concluded that pure Braitenberg vehicles are efficient but unstable for the task. In turbulent flow, a direct coupling between gradient and actuator control is likely to lead to considerable instability and often inefficient paths. However, when seeking point sources such as light, or odors under water, the strategy can be much more useful. For instance, [34] deployed a Braitenberg vehicle underwater for odor source localization, and observed behavior remarkably similar to that of lobsters. Lobsters use odor source localization by two spatially separated sensors to locate food. The biomimetic Braitenberg vehicle in [34] helped to better understand the Lobster's strategy.

Algorithm 1: Hex-path algorithm

```
while True do
    if (I_{n-2} > I_{n-1} and R_{n-1} = A.C.) or
     (I_{n-2} < I_{n-1} and R_{n-1} = C.W.) then
        rotate 60° A.C. ;
        move forward distance m ;
    end
    else
        rotate 60° C.W. ;
        move forward distance m ;
    end
end
```

Algorithm 2: E. coli algorithm

```
while True do
    if current gas_read > previous gas_read
     then
        rotate ± random(5 °) ;
        move forward random(0.05 m) ;
    else
        rotate ± random(180 °);
        move forward random(0.05 m) ;
    end
end
```

Hex-Path Algorithm

The hex-path algorithm (Figure 2.2, Algorithm 1), originates from an early feasibility study in the RoboMole project [104], that aimed to develop a robotic mole capable of navigation based on chemical information. In Algorithm 1, I_n is the measured intensity at step n and R_n is the rotation made at step n, clockwise or anti-clockwise. The strategy might initially be difficult to understand, but can be clarified

with Figure 2.2. Let's assume $I_{n-2} > I_{n-1}$, i.e. from $n-2$ to $n-1$ a negative gradient was measured. The agent should now rotate anti-clockwise at n, as rotating clockwise would move in the same negative gradient direction. In short, if a positive gradient exists between $n-2$ and $n-1$ we want to move in the same direction, whereas we want to avoid that direction if the gradient was negative.

E. coli Inspired Methods

The E. coli-inspired algorithm [81] in Algorithm 2 is more straightforward, roughly maintaining heading with positive heading while considerable changing course with negative gradient. A considerable drawback of this E. coli-inspired and the hex-path algorithm is that they heavily rely on gradient information, which is often unstable and noisy. As a consequence, they are only useful in simulator or for seeking a stable point source.

A more stable (E. coli) bacteria-inspired algorithm is the Biased Random Walk (BRW) [80]. Researchers first developed a BRW to model E. coli's behavior they observed under a microscope [10]. Every step a heading change and step length is computed, described by a Poisson distribution. After every step a new Poisson distribution is generated, imposing a longer step for increasing positive chemical gradient. The mean of the Poisson distribution describing the step length, λ_{step}, is scaled linearly with chemical gradient and steepness α.

In [79], a BRW algorithm was implemented along with a very simple chemotaxis algorithm. The chemotactic approach measures with two distinct sensors, and moves towards the direction with highest reading. The authors of [79] show that this simple strategy leads to more direct paths compared to the BRW algorithm. They conclude the BRW is more suitable for implementation on a robot as it is far more robust, but produces inefficient paths. Chemo-BRW [24] has emerged as a hybrid between the two, improving efficiency of the BRW algorithm.

2.1.3. Chemotaxis and Anemotaxis

The algorithms in Section 2.1.2 have had limited success in real-world applications, as they are either too unstable or inefficient. In nature, organisms regularly exploit wind vector information in odor source localization. This has resulted in a class of algorithms in robotics that uses chemical readings (chemotaxis) and wind information (anemotaxis). As it turns out, considering wind information leads to some of the most efficient and robust bio-inspired GSL algorithms. We review the most influential chemotactic and anemotactic strategies based on two organism: the dung beetle and silkworm moth.

Dung beetle Inspired Algorithms

The dung beetle uses a simple strategy to follow odour plumes from a cow pat [16]. It first performs a linear search (Section 2.1.1) at $\theta = 90°$, and consecutively engages in reactive plume tracking referred to as the zigzag algorithm [45].

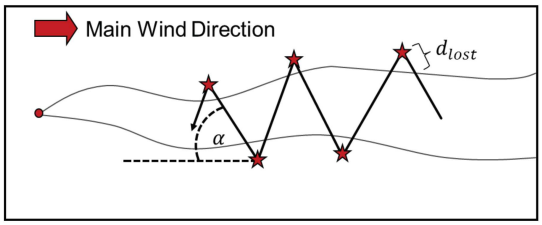


Figure 2.3: Dung beetle zigzag algorithm [45]. The agent performs a zigzag pattern at an angle α with respect to the wind vector. When it loses the plume it will continue traveling for d_{lost} before it continues the zigzag pattern. Illustration originates from [92].

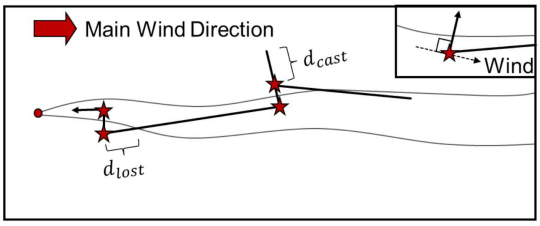


Figure 2.4: Surge-cast algorithm [75]. The agent flies upwind (surge), then swings side to side when it loses the plume (casting). The wind direction is only measured when the robot switches from strategy, surge to cast or cast to surge. In a perfect flow field all surges would be parallel, but in practice local wind vectors vary.Illustration originates from [92].

The reactive plume tracking, as shown in Figure 2.3, engages in a zizag pattern at angle α with respect to the wind vector. If the agent loses track of the plume for a distance d_{lost}, it changes heading and continues the zigzag pattern. This method was implemented on a robot in [105], where the authors constructed an environment on a table tennis table. A fan was used to spread 5% ammonia, resulting

in turbulent flow. Out of the 10 experimental runs, the source was touched 7 times, passed by closely 2 times and not found at all once, due to a spurious sensor. Compared to a Braitenberg vehicle tested in the same environment, the dung beetle algorithm is more robust but less efficient in path length. The next section considers the moth-inspired algorithms that are closely related to the zigzag approach, but result in more efficient paths.

Silkworm moth inspired algorithms

As introduced in Section 2.1.1, the male silkworm moth locates female mates by tracking a pheromone plume the female releases. Early work described the behavior solely from a biological perspective [4, 51, 52, 82, 139], and concluded the efficiency of the strategy is astonishing.

The male moth first waits before it detects pheromone (i.e., passive monitoring), it then starts flying upwind (surge). If it loses the plume it will start swinging from side to side with increasing amplitude (casting). Hence the algorithm is referred to as the surge-cast algorithm [75].

Different flavors of the algorithm are surge-spiral [38, 75] and pure casting. Surge-spiral is similar to surge-cast, only now casting is replaced by an Archimedes spiral. The robot estimates its position relative to the plume (i.e., left or right), based on wind information. It uses that information to determine the desired spiral orientation. In pure casting the robot performs a strategy very similar to the zigzag dung-beetle algorithm, only now after it has lost the plume for d_{lost}, it travels perpendicular to the wind vector until it acquires the plume. Consecutively, it continues at an angle α w.r.t. the wind vector.

Regardless of the origin of the strategy, nearly all bio-inspired chemo- and anemotaxis algorithms consist of the building blocks we have covered. By analysis of robot experiments [74, 75], simulation [72] and mathematical modelling [73], it was established that surge-cast and surge-spiral are the most effective bio-inspired strategies.

2.1.4. Real-world effectiveness

The main advantages of the bio-inspired algorithms are their low compute requirements and optimal behavior in certain environments. Mostly in open spaces and relatively stable airflow they perform well. The low computational requirement makes them a suitable for implementation onboard a nano drone, however their limited usefulness in complex environment makes real-world use challenging.

It's an interesting question if insects have evolved to search for gas in an optimal way. A key aspect to consider is that they do not only evolve a brain, but also have great sensing capability. Their sensing goes way beyond what we can achieve with sensors of this scale today. Hence, it is uncertain if the bio-inspired strategies are optimal with our hardware.

The bio-inspired algorithms might be close to optimal in nature, though an interesting research question is: *what GSL behavior is optimal on a source-seeking robot?*.

2.2. Probabilistic and map-based methods

The methods reviewed up to now are relatively simple and rule-based. Another promising class of algorithms are probabilistic, often generating a belief map with probabilities for each discrete possible source position. They do so based on an internal flow model, and differ in strategy (i.e., decisions based on that map), or flow model.

Its main observed assets are its robustness in turbulent flow and efficient multi-agent execution. The three categories reviewed here are 1) Bayesian-based strategies, 2) Hidden Markov Models, and 3) Kernel Methods.

2.2.1. Bayesian Inference

Source Term Estimation (STE)

Bayesian methods need a plume model to generate a source likelihood map (SLIM). Early methods [5, 27] assumed a Gaussian distribution in the flow direction, according to Equation 2.1 and visible in Figure 2.5. Along the axis of the plume x, parallel to the wind direction, the concentration profile can be described by a Gaussian distribution with increasing standard deviation. While Gaussian Plume Models (GPM) can be run in near real-time, their accuracy is often times low in presence of turbulent conditions and obstacles.

Gaussian plume models have been deployed repeatedly for source term estimation (STE) [12, 43]. Based on the history of recorded measurements they estimate the most likely source parameters.

$$\frac{\overline{C}}{C_m} = exp\left(\frac{-y^2}{2\sigma^2(x,F)}\right) \tag{2.1}$$

$$\mathbf{X}(t_1,t_k) = \int_{t_1}^{t_k} \mathbf{U}(\mathbf{X}(\tau))d\tau + \int_{t_1}^{t_k} \mathbf{N}(\tau)d\tau + \mathbf{X}_s \tag{2.2}$$

$$S_{ij}(t_1,t_k) = \frac{e^{-\frac{\left(x_j - x_i - v_x(t_1,t_k)\right)^2}{2(t_k-t_1)\sigma_x^2} - \frac{\left(y_j - y_i - v_y(t_1,t_k)\right)^2}{2(t_k-t_1)\sigma_y^2}}}{2\pi\,(t_k - t_1)\,\sigma_x\sigma_y} L_x L_y \tag{2.3}$$

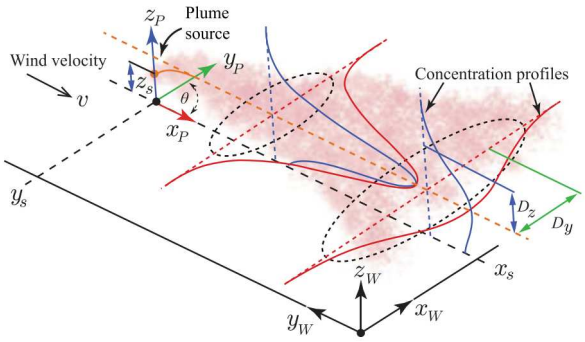


Figure 2.5: Representation of Gaussian Plume Model, from [12].

Advection-based

In [96], a novel advection-based approach was introduced. Instead of considering a time-averaged concentration, this model aims to use 'spikes' of chemical detections, especially useful in unstable flow. Equation 2.2 describes the core idea of this strategy, being to let particles of gas travel along a wind path. In Equation 2.2, $\mathbf{X}(t_1,t_k)$ represents a position (x,y) distribution for a particle at t_k, released at time t_1. $\mathbf{U}(\mathbf{X}(\tau))$ represents the position-dependent wind vector, while $\mathbf{N}(\tau)$ is a zero-mean and Gaussian random process with σ_x^2, σ_y^2 variances. $\mathbf{X}_s$ is the start position of the particle at time t_1. Assuming that the agent was alive from t_1, it will use the measured wind vector (v_x, v_y) at the agents position in that time frame. This, however, initiates an inaccuracy, as the wind vector is both space and time dependent. The $\mathbf{N}(\tau)$ term is designed to account for this inaccuracy, but it remains the single biggest weakness of this model: assuming a constant global flow field, which is especially inaccurate in presence of obstacles.

Using Bayes' theorem, Equation 2.3 can be derived. For the full derivation we direct the reader to the original work [96]. $S_{ij}(t_1,t_k)$ describes the probability of a particle detected at cell C_j detected at time t_k, originated from cell C_i at time t_1. To generate a full source probability map, we now have to compute $S_{ij}(t_1,t_k)$ for $t_1 \in [t_0,t_k]$ and for each cell. The original work [96] proposes a strategy to reduce computational effort, but the computational cost will still be considerable.

Infotaxis

In [96] the novel advection-based model was proposed to construct probability maps, but it was not coupled to robot motion. In [130], Infotaxis was introduced. In infotaxis, the agent has access to the probability map from [96] and locally maximizes information gain, hence infotaxis. It does so by computing the expected change in Shannon's entropy [110] of moving to a specific cell. It locally maximizes the reduction of entropy, and with that gained knowledge. The rationale is that cues arrive at a higher rate closer to the source, such that the agent will seek the source.

Infotaxis is probably the most well-known (probabilistic) method for gas source localization. Ever since its invention, modifications and performance analyses have been contributed [6, 37, 86, 91, 131] In [131] Infotaxis was compared to three bio-inspired algorithms in a small simulation environment, with the agent starting 2 m from the source. The authors concluded the bio-inspired algorithms are more efficient in presence of a high chemical doses, while infotaxis shows to be superior in lower concentrations. In [91], infotaxis was tested in dilute conditions in a larger 4 x 5 m area, showing that infotaxis is robust even in larger testing setups. To the best of our knowledge, no testing has been carried out in larger and more challenging environments to this date.

Hidden Markov Models

In [30], a Hidden Markov Method was adjusted for plume mapping. The hidden Markov plume model (HMPM) is represented by the parameter vector $\lambda = \left[\pi, \{\mathbf{A}(t_i)\}_{i=0}^{f}, \mathbf{b}\right]$. Here π is the source probability vector, $\mathbf{A}$ the state transition matrix and $\mathbf{b}$ the detection probability vector.

$\mathbf{A}$ represents the probability for a plume of detectable odor to move from cell $C_k(t_i)$ to $C_k(t_{i+1})$. $\mathbf{A}$ is computed by collecting the previously measured wind vector at time t_i, computing the distance traveled in dt. Now, assuming this distance is smaller than the cell size, the projected traveled distance

is normalized over cell size. This is now used as the probability for detectable odor to leave cell $C_k(t_i)$. $\mathbf{b}$ is the probability of detecting an odor, given that the current cell contains odor. The elements of $\mathbf{b}$ are identical, as the sensor performance is assumed to be constant.

Based on both $\mathbf{A}$ and $\mathbf{b}$, the agent updates the source probability vector (map). Contrary to infotaxis, the HMPM follows the most likely paths. In the original work [30] only simulation experiments were demonstrated, while in [96] the model was tested on a dataset. To the best of our knowledge, no full-fledged robot experiments have been performed with a HMPM.

The main weakness of this strategy is, similar to the advection-based model from [96], that the flow is assumed to be spatially invariant. In an open space this may not be a problem, but in an environment with numerous obstacles this may become catastrophic. There is no apparent solution to this problem, as the agent only has access to its own wind vector information.

2.2.2. Gas Distribution Mapping (GDM)

Reactive searching for a gas source has the potential to be the most efficient strategy for odour source localization. However, especially in early works, gas was often simply mapped along a scanning pattern. The first work exploiting this idea [67] simply created a gridmap with the average of the measured gas concentration. We review the existing GDM work, as it may be a practical solution in emergency response.

Kernel DM+V

The Kernel DM+V algorithm was proposed in [66], having the advantage of not relying on a particular plume model or environmental conditions. Kernel DM+V is a non-parametric estimation approach, interpreting measurements as noisy samples from the distribution that is to be estimated. The goal is to learn a distribution of gas measurements as a function of position.

To learn this distribution, it needs a trajectory that roughly covers the space, though the coverage doesn't need to be even. The measurements are weighted using an uni-variate Gaussian weighting function to represent the importance of a measurement, making measurements more important closer to the center of the cell. It then approximates the mean and variability (not variance) of gas readings in each cell. A Bayesian solution would be able to arrive at the covariance of the mean of the distribution, while Kernel DM+V carries out two parallel estimation processes to estimate variability and mean independently.

[66] demonstrated various indoors and outoors robot experiments, arriving at a high-quality gas distribution map. Kernel DM+V adds a useful kernel approach to GDM, but should be elaborated on with more sophisticated and efficient scanning patterns. It may be a useful tool in emergency operation, but its biggest weakness is the requirement to roughly cover the entire searching area, before obtaining a reliable source probability map.

SLAM-GDM

Simultaneous Localization and Mapping (SLAM) has proven to be an effective approach for complex exploratory tasks in robotics [55]. SLAM-GDM [50, 117] is a hybrid between Kernel DM+V and Hector SLAM [56]. The main contribution in SLAM-GDM is that it can now perform Kernel DM+V without a positioning system, SLAM-GDM merges the SLAM occupancy map with the gas distribution map. The entire systems runs in real-time and is controlled from a base station, making it a practical solution for emergency response. Though fully autonomous operation, especially in swarm configuration, is likely to be more efficient.

2.3. Source declaration

Source declaration adds up to identifying a source when the robot is within proximity. In many algorithms this stage is not specified, and is generally carried out by camera or human vision.

In the zigzag algorithm [45](Section 2.1.3), the agent analyzes the spatial proximity of plume detection points. If the points are far apart, this indicates a distant source, assuming the plume has already been acquired (i.e., the robot is situated down-wind w.r.t. the plume). One of the challenges with this strategy is that it requires human insight to determine determine a threshold for detection proximity, which is source-dependent.

In the surge-spiral algorithm [38, 75](Section 2.1.3), agents climb up-wind with their surge-spiral strategy, until they have established a pattern. If the agent reaches the source, and surges even more

upwind, it will lose the source and start spiraling until it has found the source again. This manoeuvre will repeat indefinitely, making it obvious to detect a pattern. While it might be possible to develop a source-independent implementation of this strategy, it's weakness is that local optima might look like a source. A small cycle of short surges and casts may occur even at distance from the source.

In [68] local optima were used to predict source proximity, where a higher frequency of local optima signifies closer proximity to the source. Similar to the zigzag's declaration approach, this strategy has the issue that a human threshold needs to be defined. In [67] a gas concentration map was used to estimate the source position, based on the cell of the highest concentration. [65] argues a series of measurements is required for source declaration. It performs a number of turns and subsequently feeds the measurements into a support vector machine (SVM) [17].

A few map-based and probabilistic methods have been exploited to declare source position. [93] proposed a particle filter-based approach to source declaration.

3

Multi-Agent Gas Source Localization

In a real emergency response, the objective is to localize the gas source efficient and reliably. While a single robot might be capable of the task, it is likely that deployment of multiple robots, especially when coordinated, will lead to better results. In the best case, the agents will collaborate and share information, leading to more efficient paths [76]. If a group of robots is deployed without collaboration but with physical interaction, the performance gain will be limited.

We divide this chapter in traditional robot swarming and bio-inspired algorithms (Section 3.1) and Probabilistic and Map-based strategies (Section 3.2).

3.1. Traditional Robot Swarming and Bio-Inspired Algorithms

3.1.1. Chemotaxis

Gradient-Based

Perhaps the most simple gradient-based multi-agent chemotaxis algorithm was proposed in [83], which uses a local localization system communicates between agents. Out of the visible agents, each agent moves towards an agent randomly picked from those with highest reported gas intensity. The authors provide robot experiments, but do not show a comparison or exhaustive review of performance.

Particle Swarm Optimization (PSO)

Particle swarm optimization (PSO) is a computational optimization method, inspired by social behavior of animals such as bird flocks and fish schools [54]. PSO is a central strategy in multi-agent GSL, we therefore review the fundamental concepts of the algorithm.

The original PSO algorithm is displayed in algorithm 3. Each agent makes a decision based on 1) the best position it has seen, 2) the best position the swarm has seen and 3) uniform random variables. S agents are initialized with a uniformly distributed initial position and velocity, $\mathbf{x}_i \sim U\left(\mathbf{b}_{l0}, \mathbf{b}_{\mathrm{up}}\right)$ and $\mathbf{v}_{\mathrm{i}} \sim U\left(-\left|\mathbf{b}_{\mathrm{up}}-\mathbf{b}_{\mathrm{lo}}\right|, \quad\left|\mathbf{b}_{\mathrm{up}}-\mathbf{b}_{\mathrm{lo}}\right|\right)$. Here b_{up} and b_{lo} are the boundaries of the search-space. Then, in each step, the new velocity of the agent is determined based on the position of the best found position by the swarm, and the best position the individual has seen. ω, ϕ_p and ϕ_g are hyper parameters configured by the user.

In [49] PSO was first implemented for GSL, demonstrating different approaches to PSO-based GSL, proposing both chemotaxis and anemotaxis strategies based on PSO (Subsection 3.1.2). [106] first demonstrated feasibility in the real world, by deploying three ground robots in a 4.88 x 4.88 m test area. The bare form of PSO resulted in a success rate of 4/6 experiments with an average search time to success of 252 seconds. While the experiments proved feasibility, success rate and efficiently can be improved on.

A substantial weakness of PSO-based algorithms is that they assume the global and local optimum to be constant, i.e. the stagnation assumption [11]. In the context of GSL this assumption is invalid, as the concentration map of the environment is time-variant. A possible solution to this problem was proposed as Detection and Responding PSO (DR PSO) [49]. In DR PSO, if the global optimum has not changed for a number of iterations, the agents are randomly spread across the environment. This is especially relevant in the plume acquisition stage, when local optima are frequently observed. Finally,

in [97], Force Field Particle Swarm Optimisation (FFPSO) was proposed such as to avoid collisions with other agents. It was implemented on a swarm of Crazyflie nano drones [32], though in a small area with a simulated source.

Algorithm 3: PSO Algorithm

for *each particle* $i = 1, \dots, S$ **do**
  Initialize the particle's position with a uniformly distributed random vector: $\mathbf{x}_i \sim U\left(\mathbf{b}_{l0}, \mathbf{b}_{\mathrm{up}}\right)$;
  Initialize the particle's best known position to its initial position: $\mathbf{p}_i \leftarrow \mathbf{x}_i$;
  if $f\left(\mathbf{p}_i\right) < f(\mathbf{g})$ **then**
    update the swarm's best known position: $\mathbf{g} \leftarrow \mathbf{p}_i$;
  end
  Initialize the particle's velocity: $\mathbf{v}_{\mathrm{i}} \sim U\left(-\left|\mathbf{b}_{\mathrm{up}} - \mathbf{b}_{\mathrm{lo}}\right|, \quad \left|\mathbf{b}_{\mathrm{up}} - \mathbf{b}_{\mathrm{lo}}\right|\right)$;
  while *a termination criterion is not met* **do**
    for *each particle* $i = 1, \dots, S$ **do**
      for *each dimension* $d = 1, \dots, n$ **do**
        Pick random numbers: $r_p, r_g \sim U(0,1)$;
        Update the particle's velocity:
        $\mathbf{v}_{i,\mathrm{d}} \leftarrow \omega \cdot \mathbf{v}_{1,\mathrm{d}} + \varphi_{\mathrm{p}} \cdot r_{\mathrm{p}} \cdot \left(\mathbf{p}_{1,\mathrm{d}} - \mathbf{x}_{\mathrm{i,d}}\right) + \varphi_{\mathrm{g}} \cdot r_{\mathrm{g}} \cdot \left(\mathbf{g}_{\mathrm{d}} - \mathbf{x}_{\mathrm{i,d}}\right)$;
      end
      Update the particle's position: $x_i \leftarrow x_i + v_i$;
      if $f\left(\mathbf{x}_{\mathrm{i}}\right) < f\left(\mathbf{p}_{\mathrm{i}}\right)$ **then**
        Update the particle's best known position: $\mathbf{p}_{\mathrm{i}} \leftarrow \mathbf{x}_{\mathrm{i}}$;
        if $f\left(\mathbf{p}_i\right) < f(\mathbf{g})$ **then**
          Update the swarm's best known position: $\mathbf{g} \leftarrow \mathbf{p}_{\mathbf{i}}$;
        end
      end
    end
  end
end

Biasing Expansion Swarm Approaches (BESA)
A Biasing Expansion Swarm Approach (BESA) for GSL was proposed in [18]. The algorithm divides the search area into a discrete grid, with no more than a single agent in each grid cell. Then, in each step, each robot in the swarm can expand to a cell that has at least one agent in its adjacent cells, to keep the swarm together.

For all expansion cells, each cell' biasing parameter B is computed by $B_{(x,y)} = \frac{K}{n} \sum_{i=0}^{n} \left(\frac{C_i}{r_i^2}\right)$. Here K is a constant, n the number of agents contributing their readings to the swarm, C_i the concentration agent i is recording, and r_i is the distance from (x, y) to the position of agent i. Intuitively, an agent' reading contributes more when its closer to the expansion cell considered, similar for a higher reported concentration. Each agent makes a choice based on the cell with the highest biasing parameter.

Simulation results were presented in [18], demonstrating BESA is more efficient as compared to a gradient-seeking baseline, with no collaboration. To the best of our knowledge, no physical robot experiments exist.

Glowworm Swarm Optimization (GSO)
GSO [58] is a swarm intelligence and optimization algorithm inspired by glowworms, also referred to as fireflies or lightning bugs. In GSO, the 'fireflies', glow at an intensity that approximates the fitness function at that point in space. Flies with higher intensity attract flies with lower intensity. It also involves a dynamic decision range, discounting fireflies that are far away, especially when plenty other closer agents are visible. This makes the strategy different compared to evolutionary multi-modal optimization [33], and makes GSO capable of localizing multiple local optima. This is of great interest in GSL, as local optima regularly exist in presence of turbulence and obstacles.

In [57], simulation results are presented for odor source localization by GSO. It also includes a simple obstacle avoidance model by proximity-detection sensors. In simulation the swarm was successful, being particularly suitable for locating multiple sources. From using 13 agents, on average 2/3

sources were located. However, the sources were modeled as dispersion, which isn't considered to be accurate in many real-world applications. The work was extended by phototaxis robot experiments, implementing GSO on a swarm of ground robots [58].

A modified GSO (M-GSO) was applied in a simulation environment in [144], with its main feature being a global random search of self-exploration and local search based on GSO. The objective of M-GSO is to achieve superior exploration. The author's simulations show M-GSO outperforms GSO, finding 10/10 local optima compared to 7/10 by GSO.

Biased Random Walk (BRW)
In Section 2.1.2 BRW was introduced as a relatively robust method, but without a measure to allow for communication. [80] proposed computation of a bias to allow for collaboration, as shown in Equation 3.1.

$$\text{Total\_ Bias}_i = \frac{\sum_{j=1}^{N} \alpha_{i,j} \times RP_{i,j}}{\text{No\_agents}} \text{ for } i, j, 2, \dots, N \text{ and } i \neq j \tag{3.1}$$

Here Total_ Bias_i is the bias angle added to the locally computed angle from local measurements. $RP_{i,j}$ is the angle agent i has to rotate to face agent j. $\alpha_{i,j}$ is computed by $\alpha_{i,j} = f_j / \left(f_i + f_j\right)$, where f_j is the concentration measurement of agent j. Hence, agent i will be biased towards agents with high chemical readings.

The authors of [80] applied the bias term to both BRW and Chemo-BRW in simulation, deploying a Fire Dynamics Simulator (FDS), based on Computational fluid Dynamics (Section 5). They found substantial efficiency improvements when collaborating through the bias term, with more significant improvement for BRW (∼20 % shorter paths) as compared to Chemo-BRW (∼10 % shorter paths). This is likely caused by the superior performance of single-agent Chemo-BRW, having less room for improvement by collaboration. To the best of our knowledge, no robot experiments tested this approach.

3.1.2. Chemotaxis and Anemotaxis

PSO-Based
The first PSO-based GSL work [49] also contributed modified PSO (MPSO), incorporating wind information. The authors propose two distinct MPSO algorithms: wind utilization implementation I, WUI1, and implementation 2, WUI2.

In WUI1 two vectors are constructed, being $\mathbf{V}_i^*$, the output from the bare PSO algorithm, and $\mathbf{W}(t)$, the local wind vector. The angle between the two vectors is defined as θ. In WUI1 the final moving vector, $\mathbf{V}_i$, is equal to $\mathbf{V}_i^*$, expect for the case when θ is between a certain range, in that case $\mathbf{V}_i = 0$. This range is also referred to as a forbidden area, preventing the agent to go downwind. A weakness of this approach is that the forbidden range needs to be carefully configured, having a great impact on performance.

In WUI2, the same two vectors are used to construct $\chi_\theta \left(\mathbf{W}(t), \mathbf{V}_i^*(t)\right) = \frac{1}{2}\left(1 - \left(\mathbf{W}(t), \mathbf{V}_i^*(t)\right)\right)$. χ_θ is a sinusoidal function with $\chi_\theta = 0$ for $\theta \in [0, 2\pi]$ and $\chi_\theta = 1$ for $\theta = \pi$. Now, the final computed velocity vector can be computed by $\mathbf{V}_i = \chi_\theta \mathbf{V}_i^*$. Taking a step back, this means that if the original PSO outputs a velocity vector that is completely downwind, $\mathbf{V}_i = 0$. On the other hand, if the original PSO sends the agent straight up-wind, $\mathbf{V}_i = \mathbf{V}_i^*$.

Ever since its first introduction, PSO has been applied in a variety of robot experiments. While most of them solely consider a small-scale and obstacle-free scenario, the authors of [125] execute a realistic set of experiments. The authors released four 20 lb propane tanks, in a $\sim 35x40m$ outdoor area. A more practical 'waypoint-PSO' was introduced, updating the next waypoint occurs only if all agents have reached their next waypoint, or run out of time. The Enif quadcopter used in these experiments [39] is equipped with a lidar sensor to avoid obstacles in its trajectory by the open sector collision avoidance method [126]. Another PSO-based algorithm modified for GSL is Probability-PSO [64] (P-PSO). P-PSO uses bayesian inference and variable-universe fuzzy inference to estimate source location probability. The idea here is that chemical readings are unstable, hence using a source likelihood map may yield better results. All agents share a merged likelihood map. The work only contains high-level simulation experiments, making it hard to judge its performance.

Extending Bio-Inspired Algorithms
An interesting question is how a swarm of agents perform when each individual is loaded with a bio-inspired strategy. In [76], the authors compare deployment of a swarm of robots individually performing

casting, surge-spiral and surge-cast. In a set of simulation experiments, up to 5 robots are deployed in a 18 m x 4 m arena, avoiding the other agents by a Braitenberg [13] obstacle avoidance strategy. The authors show that swarm deployment of the single-agent bio-inspired algorithms does not yield significant performance improvement. Especially for casting and surge-casting, the lowest traveled distance before source localization is not significantly improved when deploying a swarm.

Only when deploying the surge-spiral algorithm, especially with a wider spiral, significant improvement is observed in performance. The spiral makes the robot leave the plume for a substantial time, giving the other robots time to pass (i.e., less inference). Unfortunately, this surge-spiral configuration is inefficient, making deployment of single-agent bio-inspired algorithms without cooperation inefficient. Naturally, the entire system becomes more robust to failure of a single agent, but efficiency can be improved only by cooperation.

Two cooperative versions of spiral-surge were presented in [38], `attract` and `kill`. In `attract`, a robot that has found a plume signal sends a 'come here' signal to all other agents, to point them in the right direction. In `kill`, once one agent reaches a certain chemical threshold, all other agents are put into a power-saving mode. The purpose of this strategy is to develop a swarm capable of locating a gas source before all batteries run out. The authors demonstrated, in a range of robot and simulation experiments, that `attract` and `kill` outperform a non-cooperative swarm of spiral-surge robots. The strategies are shown to gain advantage over no communication with an increasing group size, especially when deploying more than five robots.

Formation-based swarming

Formation-based control is a well-known strategy in robot swarming [15]. The agents align in a certain formation while completing their objective, maintaining communication. Over the past decades, different formation-based strategies were developed for GSL [20, 77, 113–115]. The first formation-based GSL strategy[77] applies an upwind force f_u and a crosswind force f_c to each agent. While the wind vector may differ from agent to agent, it is an approximation of a global coordinate system between all agents.

$$f_u = u + \frac{1}{N}\sum_i y_i \tag{3.2}$$

$$f_c = af_a - rf_r \tag{3.3}$$

$$f_a = \frac{\sum_i x_i c_i}{\sum_i c_i} \tag{3.4}$$

$$f_r = \frac{1}{N}\sum_{i, i\neq \mathrm{me}} \frac{1}{x_i} \tag{3.5}$$

Here f_u, defined by Equation 3.2, is the force each agent applies in upwind direction. u is the constant upwind drag, N the number of agents, and y_i the location of the other agents i along the upwind axes. f_u keeps the agents aligned along the wind axis, an agent that is running behind will see larger y_i's and resulting in a larger upwind force.

f_c (Equation 3.3), is the crosswind force and a weighted difference between an attractive and repulsive force. The attractive force f_a draws agents towards other agents with high gas concentration readings c_i. x_i is defined as their position on the crosswind axis. Finally, the repulsive force f_r is put in place to maintain distance between the individuals, i.e. stay in formation.

Summarizing, in wind direction this algorithm keeps all agents aligned while in crosswind direction it seeks superior gas concentration. The authors tested the strategy with up to five robots in a small wind tunnel, and demonstrated performance superior to previous bio-inspired algorithm. The algorithm should be tested in more complex and turbulent environments, as a wind tunnel provides a relatively stable flow field.

After this first formation-based GSL strategy, novel works contributed additional theories and experiments. In [85] the authors investigated the effect of different behaviors. Their simulations and experiments confirmed that a crosswind formation is optimal, especially when the agents are equidistant with respect to each other. The optimal distance between agents depends on wind speed. Around the same time, [113] demonstrated alternative formation configurations. The experiment set was very

small, making these impossible to be compared to a cross-wind formation. To the best of our knowledge, no formation-based GSL algorithms have been applied in larger and more complex environments.

Fluxotaxis

Fluxotaxis [142] is a framework for physics- and formation-based control of a swarm of gas-seeking robots. It allows a fully autonomous swarm to self-organize into a structured lattice arrangement, maintaining the formation even when avoiding obstacles.

The lattice between the robots is maintained by a force law, introducing either attraction or repulsion. The force is defined by $F = \frac{G}{r^p}$, where F is the force magnitude, G a constant, r the distance between two agents, and p another user-defined constant. The desired separation between two robots is R, making F repulsive if $r < R$ and attractive when $r > R$. The lattice can contract and expand by varying R and take different shapes.

The idea behind fluxotaxis is to not seek a local chemical reading, but to seek mass flux: $\rho\vec{V}$. Equation 3.6 describes the divergence of mass flux, which is positive for a source. Here ρ is the chemical density of the gas to be located, $\vec{V}$ the velocity vector, u the x component of velocity and v the y component.

$$\nabla \cdot (\rho\vec{V}) = u\frac{\partial \rho}{\partial x} + \rho\frac{\partial u}{\partial x} + v\frac{\partial \rho}{\partial y} + \rho\frac{\partial v}{\partial y} \tag{3.6}$$

Each agent in the lattice computes the local amount of chemical flux, $\rho\vec{V}$. The final applied force vector of each agent is the sum of the attraction-repulsion force and a force in the chemical flux direction. Hence, the lattice will be drawn towards agents detecting more mass flux. If very low mass flux is detected by the swarm, it will resort to an additional approach to maximize use of available information, as described in [118].

The authors [142] test the lattice repulsion-attraction strategy not only with their mass flux strategy, but also with chemotaxis and anemotaxis strategies. In chemotaxis the lattice-introduced force is summed with a force in the direction of individually recorded chemical gradient, while in anemotaxis every individual agent performs a surge-cast algorithm [75], while keeping the lattice in formation.

The authors show a very convincing win for fluxotaxis, in both simulation and robot experiments. A shortcoming of this work is that the strategies are highly susceptible to parameter tuning. For instance, in the chemotaxis strategy, the authors do not expand on how the final chemical force is computed. Let's say a gradient $\frac{\partial C}{\partial t}$ is observed, what force is applied to the individual? Without knowing the maximum expected $\frac{\partial C}{\partial t}$ over a trial run, answering this question is difficult. Seeing such a convincing win for fluxotaxis, we should ask the question if all strategies are configured correctly. Fluxotaxis certainly is an interesting idea though, as it merges anemotaxis and chemotaxis information.

Frontier-Based

In [84] a frontier-based approach to GSL was proposed. A frontier is the edge of the explored area, that is not a wall or obstacle (Figure 3.1). The swarm of robots may decide to explore one of the frontiers

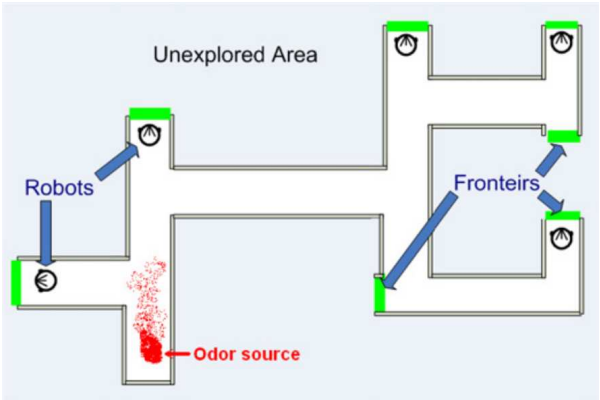


Figure 3.1: Problem description from [84], frontier based GSL.

based on a cost and utility function. Each agent travels to the frontier that has highest profit, where $profit = utility - \beta \cdot cost$. In this case cost is proportional to required distance traveled and utility is odor concentration at the frontier.

The system is decentralized, performing map merging for obstacles and odor measurements through topological graph matching [41]. The agents also correct measurements by overwriting previous readings if they move in an already explored area. This is useful in light of the time-variant nature of gas sources. The authors tested the approach in simulation and in real experiments with three robots in a small but complex environment.

3.2. Probabilistic and Map-based

This section reviews the contributions in multi-agent probabilistic and map-based strategies for GSL. An essential distinction between the different works is the way source likelihood maps are merged.

Infotaxis

Infotaxis [130] might be the most well-known probabilistic GSL strategy. Many works have expanded on the original infotaxis, expanding the algorithm to work in a multi-agent setting. In [101], all-to-all communication is assumed to accomplish a centralised fusion architecture. In other words, all data is used to generate a common likelihood map. By a series of simulation experiments, the authors show a diminish of return in mean search time after deploying more than seven platforms, and show a linear growth of the mean search time with the search area.

One of the problems with assuming all-to-all communication is that relatively large amounts of data are moved around, which can slow down the algorithm. A recent contribution [116] first used a particle filter to generate a source probability distribution for each individual, after which a Gaussian density function is fit. Now the agents only communicate the mean and covariance matrix of the Gaussian distribution, instead of all recorded measurements. This strategy drastically reduces the required communication bandwidth, but also saves on computational effort, as each agent now only processes its own readings. Each agents merges its own map with those of its peers by the computationally cheap KL divergence [21]. They also introduce seeking 'free energy' (Equation 3.7), instead of pure entropy.

$$F_k^i = W_k^i + T \cdot S_k^i \tag{3.7}$$

Here W_k^i is the potential energy, defined by $\left\| r_k^i - \sum_{m=1}^{M} w_k^{i,(m)} r_{0,k}^{i,(m)} \right\|$, r_k^i being the current robot position and $\sum_{m=1}^{M} w_k^{i,(m)} r_{0,k}^{i,(m)}$ the estimated source position. T is the temperature that controls the relative value between the two previous terms. The main difference with pure infotaxis (i.e., seeking just S), is that we now have a multi-objective optimization problem. On one hand gaining information as fast as possible, and on the other hand traveling in the direction of highest source probability. The authors show robust behavior in simulation with three robots. The 'free energy' method outperforms pure infotaxis in efficiency in simulation.

Bio inspiration

A recent work [12] contributed a hybrid consisting of bio-inspired and probabilistic methods. The first step of the algorithm performs source term estimation (STE), fitting a Gaussian plume model with a particle filter. Once a plume model is fitted, motion planning is done with bio-inspired methods such as a Biased-Random-Walk and Surge-Cast algorithm. Real-world experiments are presented with three robots, in 4 x 4 m testing environment. While the agents seem to get closer to the source over time, it is unclear how efficient the approach is. It takes more than three minutes to locate the source within 1 meter, in a 4 x 4 m arena. It is likely that a pure bio-inspired or infotaxis approach will outperform this work.

4

Reinforcement Learning

In the field of computer vision deep learning has taken off over the past decade, thanks to its unparalleled performance in pattern recognition. Predominantly through supervised learning, deep neural networks (DNN's) have learned to estimate depth from images, recognize objects or decode handwriting. Likewise, natural language processing (NLP) has greatly benefited from supervised deep learning, powering Google's impressive live translation algorithms [138].

While supervised learning may be an excellent solution for pattern recognition, it can be impractical and non-optimal in robotics. A machine learning strategy known as reinforcement learning is an emerging solution for robot control. Instead of fitting a neural network based on user-defined labels, the agent considers the environment as a black box and trains the networks on a trial and error basis. Reinforcement learning has the potential to find the optimal solution to a given black-box problem, while by supervised learning, an agent will never get better than its labels.

We first discuss two categories of reinforcement learning, being 1) gradient-based (Section 4.1) RL and 2) neuroevolution (Section 4.2), learning through an evolutionary algorithm. Gradient-based learning assigns rewards based on the current state within an episode, while neuroevolution assess at least one episode to compute a reward. We then dive deeper into multi-agent reinforcement learning (Section 4.3) and finally review contributions in the intersection of GSL and reinforcement learning (Section 4.4). This chapter is a comprehensive review of relevant methods, and does not cover all the topics in reinforcement learning. Please refer to dedicated reinforcement learning surveys for a more thorough review of RL methods [109, 124].

4.1. Gradient-Based Learning

The most common way of training neural networks in reinforcement learning is by gradient-based optimization. The weights in the network are optimized through a stochastic gradient-descent, converging to a local or global optima. The immense popularity of gradient-based policy optimization is fed by the high-dimensional image inputs, resulting in sometimes millions of weights. While training by evolution (neuroevolution, Section 4.2), may lead to superior exploration of the solution space, many researchers think it's impractical for large networks. Hence, in this section we review recent gradient-based reinforcement learning work.

A RL algorithm optimizes a reward R, where $R = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$, the discounted future reward from $t = 0$ to $t = T - 1$. Here γ is the discount factor, with $\gamma \in [0, 1]$, and r_{t+1} the immediate reward at time step $t + 1$. T is the length of the episode. The discounted future reward is a weighted sum of all instantaneous rewards, giving less weight to more distant immediate rewards.

The immediate reward, $\mathcal{R}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$, is a user-defined function of action and states. For instance, when learning to avoid obstacles, it can be the distance to the closest obstacle. Through the discount factor, the agent can learn to optimize for a sequence of actions. Maybe the agent receives a penalty only when getting very close to an obstacle. Through the discount factor it can learn the result of sequential actions, i.e., predict moving towards a wall will eventually lead to a penalty.

4.1.1. Value Functions and Quality Functions

As we are now familiar with the problem definition in reinforcement learning, we can compare the different principles behind some of the key gradient-based algorithms. Given a policy π, the value function $V^{\pi}(\mathbf{s})$ is the expected reward when arriving in state $\mathbf{s}$. It is the expected discounted future reward given a state $\mathbf{s}$ and policy π (Equation 4.1).

$$V^{\pi}(\mathbf{s}) = \mathbb{E}[R|\mathbf{s}, \pi] \tag{4.1}$$

If the agent could reliably predict the next state based on its actions, it could generate an action as to optimize the value function. In practice, however, the dynamics of the system and environment, described by transition matrix $\mathcal{T}$, are unavailable. We need a function capable of using the current state and action, the quality function $Q^{\pi}(\mathbf{s}, \mathbf{a})$ (Equation 4.2).

$$Q^{\pi}(\mathbf{s}, \mathbf{a}) = \mathbb{E}[R|\mathbf{s}, \mathbf{a}, \pi] \tag{4.2}$$

The agent considers all possible actions and takes the action with highest expected reward.

4.1.2. Dynamic Programming

While in deep-rl the Q-function may be learned by a DNN, this is not necessarily the case. In fact, reinforcement learning is much older than neural networks, often using look-up tables for a certain state [127]. With increasing interest in high-dimensional inputs, such as camera's, learning-based reinforcement learning has gained traction. The quality function can be updated using dynamic programming.

We define the quality function in a recursive form, as in Equation 4.3.

$$Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\mathbf{s}_{t+1}}\left[r_{t+1} + \gamma Q^{\pi}(\mathbf{s}_{t+1}, \pi(\mathbf{s}_{t+1}))\right] \tag{4.3}$$

$Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ can be described in terms of $Q^{\pi}(\mathbf{s}_{t+1}, \pi(\mathbf{s}_{t+1}))$ and r_{t+1}, which will be known at the next time-step. In other words, we can generate a new quality function estimate each time step, correcting our previous estimates. $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ is updated through Equation 4.4, α being the learning rate and δ the temporal difference (TD) error.

$$Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) + \alpha\delta \tag{4.4}$$

The definition of temporal difference varies for different RL algorithms, but can generally be defined as $\delta = Y - Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$, with a different Y in different algorithms. In SARSA [103], $Y = r_t + \gamma Q^{\pi}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})$, substituting in Equation 4.4 gives Equation 4.5. The higher the learning rate α, the more the quality function will be corrected based on our current observations.

$$Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) \leftarrow (1 - \alpha) \cdot Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) + \alpha \cdot (r_t + \gamma Q^{\pi}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})) \tag{4.5}$$

SARSA is an on-policy learning method, creating a new label each time step. By contrast, deep q-learning is off-policy. It may use a replay buffer to learn $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ based on experience, setting $Y = r_t + \gamma \max_{\mathbf{a}} Q^{\pi}(\mathbf{s}_{t+1}, \mathbf{a})$.

4.1.3. Policy Gradient

Other methods are based on the policy gradient, using it to optimize by gradient ascent. First, the reward for a specific policy π, parameterized by θ, is given by Equation 4.6.

$$J(\theta) = \sum_{s \in \mathcal{S}} d^{\pi}(s) V^{\pi}(s) = \sum_{s \in \mathcal{S}} d^{\pi}(s) \sum_{a \in \mathcal{A}} \pi_{\theta}(a|s) Q^{\pi}(s, a) \tag{4.6}$$

Here $d^{\pi}(s)$ is the probability for the agent with policy π to reach state s at any given point in time. $V^{\pi}(s)$ is again the value function (Equation 4.1), making $J(\theta)$ equal to the expected immediate reward at any point in an episode. Hence it makes sense to want to maximize $J(\theta)$, as we will then maximize the expected immediate reward.

In the right part of the equation, $V^{\pi}(s)$ is replaced by the product of $\pi_{\theta}(a|s)$ and $Q^{\pi}(s, a)$. $\pi_{\theta}(a|s)$ is the probability for the stochastic policy π_{θ} to take action a given state s. $Q_{\pi}(s, a)$ is the quality function

(Equation 4.2) given we take action a from state s. To optimize $J(\theta)$ by gradient ascent we compute the gradient by Equation 4.7, derived in [128].

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[Q^\pi(s,a) \nabla_\theta \ln \pi_\theta(a|s) \right] \tag{4.7}$$

While this was the first 'vanilla' policy gradient update, many have followed to improve. This original version has zero bias, but a relatively high variance, making training less stable. Different methods were proposed to reduce variance while keeping the bias unchanged [109].

4.1.4. A brief overview of policy gradient algorithms

A large amount of policy gradient algorithms has been proposed in recent years, this section serves to highlight some of the key contributions.

REINFORCE

A great example of the theory from Section 4.1.3 is the REINFORCE [137] algorithm. According to Equation 4.8, it computes the policy parameter gradient.

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[G_t \nabla_\theta \ln \pi_\theta \left(a_t | s_t \right) \right] \tag{4.8}$$

Here G_t is the discounted future reward for time t to the end of episode at time T. Hence the computed gradient here is a local gradient at time t, that the algorithm can use to optimize for each time step. It does so by:

1. Initialize random policy parameters θ
2. Generate a trajectory with all its actions, rewards and states.
3. For each time step in the recorded trajectory, update the policy parameters using Equation 4.9 :

$$\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla_\theta \ln \pi_\theta \left(a_t | s_t \right) \tag{4.9}$$

Here G_t is known as the full episode has already been recorded, and $\nabla_\theta \ln \pi_\theta \left(a_t | s_t \right)$ can be computed through the chain-rule.

Actor-Critic

In an attempt to reduce the gradient variance in the REINFORCE algorithm, actor-critic methods learn a value function in addition to the policy. The critic attempts to learn either the quality or value function, depending on the algorithm. The actor makes actions in the environment and, guided by the critic, changes its parameters θ.

In a simple actor-critic model, Equation 4.9 is used to update the actor, only now G_t is replaced by the quality function , described by the critic.

By traveling along the trajectory, the critic can be improved too, by comparing its evaluations against the observed rewards. The critic' parameters w are updated based on Equation 4.10, where s' and a' are the state and action at step $t+1$.

$$\begin{aligned} \delta_t &= r_t + \gamma Q_w \left(s', a' \right) - Q_w(s,a) \\ w &\leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s,a) \end{aligned} \tag{4.10}$$

Other Methods

In the interest of conciseness, here we group some other relevant gradient-based algorithms. A big other category of algorithms is off-policy, which can be using any past experience to learn (i.e., learn from replay buffer). The replay buffer contains actions taken by an older policy, which may improve exploration.

Other relevant and well-known algorithms are A3C and A2C [89], DPG [111], DDPG [69], D4PG [7], TRPO and PPO [108], ACER [132], SAC [36], TD3 [31], SVPG [71] and IMPALA [25]. An exhaustive review of all these gradient-based RL algorithms can be found at [134].

4.1.5. Performance of Gradient-Based RL

Gradient-based reinforcement learning has been particularly useful in computer vision-based robot control. End-to-end control by a visual input, i.e., returning an action based on visual input, is a high dimensional optimization problem. Some of the networks have millions of parameters, which is the primary reason for gradient-based RL in the field. Evolutionary robotics approaches for deep neural networks are emerging, but have generally resulted in limited success over the past decade.

Different game environments are used to benchmark the performance of RL algorithms. Possibly the most famous are the Atari environments [88] (Figure 4.1). From the raw, high-dimensional, sensor input, the agent learns to optimize the score in the game. Some environments can be solved, reaching the last level for instance, while others have an infinite maximum score.

Most of the progress in RL has been achieved in simulation environments. RL algorithms are powerful in simulation, though their performance in real experiments is limited due to the sim2real gap. Especially end-to-end control by visual inputs is challenging. An area where RL has been really successful is low-level control, replacing optimal control strategies.

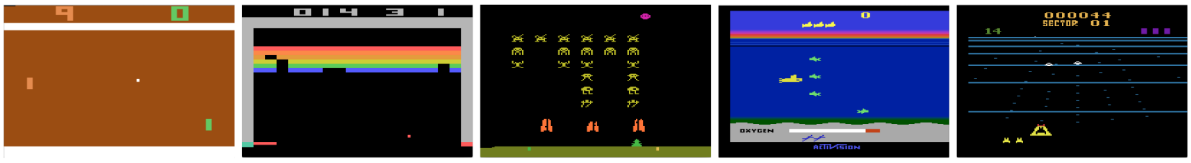


Figure 4.1: Atari game environments [88].

4.2. Neuroevolution

Neuroevolution has key properties that are unavailable to gradient-based RL algorithms. In the gradient-free neuroevolution, a population of solutions is optimized through evolutionary algorithms. The primary advantages over gradient-based optimization are greater exploration of the solution space and massive parallelization. As considered in Section 4.1, a disadvantage is that classic neuroevolution performs poorly with very large networks.

4.2.1. Classic Neuroevolution

The inspiration for neuroevolution comes from nature, where creatures evolve over generations. Genetic algorithms (GA's) are deployed in classic neuroevolution, with the goal to optimize the weights of the network. The vector containing all weights of the network is also referred to as decision vector in this optimization problem.

Genetic Algorithms

A genetic algorithm can optimize a neural network according to the following steps:

1. Evaluation: all decision vectors in the population are evaluated over the same conditions, arriving at a score for each decision vector.

2. Selection: the decision vectors are ordered in ascending order, based on fitness (if higher is better). Out of a population with size μ, λ individuals are selected for reproduction. Some GA's simply chose the top λ vectors for reproduction, though a roulette scheme is fairly common too. In the roulette case, the probability for each vector i to survive is equal to: $P(i) = \frac{C(i)-\min(C)}{\sum_{j=1}^{\mu}(C(j)-\min(C))}$.

3. Crossover: Now λ individuals are left after selection, which will be parents for the next generation. This 'mating' between parents, is known as crossover. Different crossover methods are known, such as k-point crossover, uniform crossover or exponential crossover.

4. Mutation: Before concluding this generation, all decision vectors are mutated. Again, a variety of mutation strategies has been proposed, with Gaussian mutations as a popular choice for neuroevolution. For each gene, with a probability P_m, a Gaussian random value is added to the gene.

As is hopefully clear now, in neuroevolution actions are not necessarily coupled to a reward. By contrast, at least an entire episode is evaluated on fitness, but usually even more episodes are compared for a more thorough evaluation. Even such a simple GA can optimize a neural network with impressive results.

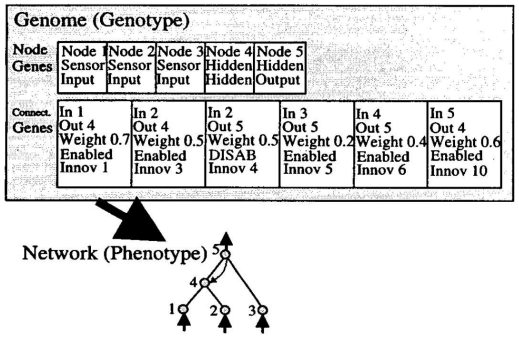
NEAT

An additional advantage of neuroevolution is that the optimization space is not limited to the weights of the network, the topography can be optimized too. An algorithm known as NeuroEvolution of Augmenting Topologies (NEAT [121]). NEAT proposed a method to crossover different topologies, as it was previously unknown how to do so.

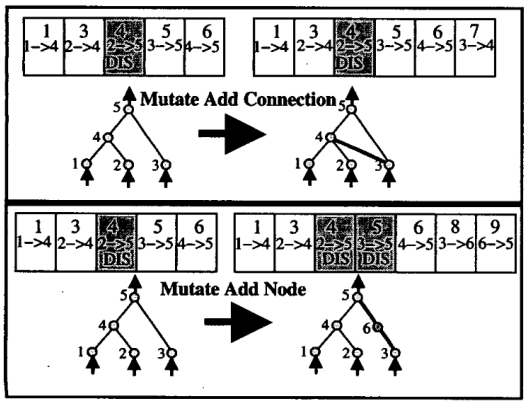
We leave the full derivation of NEAT for the interested reader in [121], but provide a brief overview of the key principles. Similar to the equivalent in biology, in neuroevolution, a distinction is made between phenotype and genotype. A genotype is the genetic representation of a network, the DNA for a creature. At the same time, the phenotype is a physical description of the creature, or in neuroevolution, the actual network.

So far we have assumed a constant network topography, making the genotype a vector containing all weight in the network. Now that the topography is varied too, the genotype needs to contain the topography too.

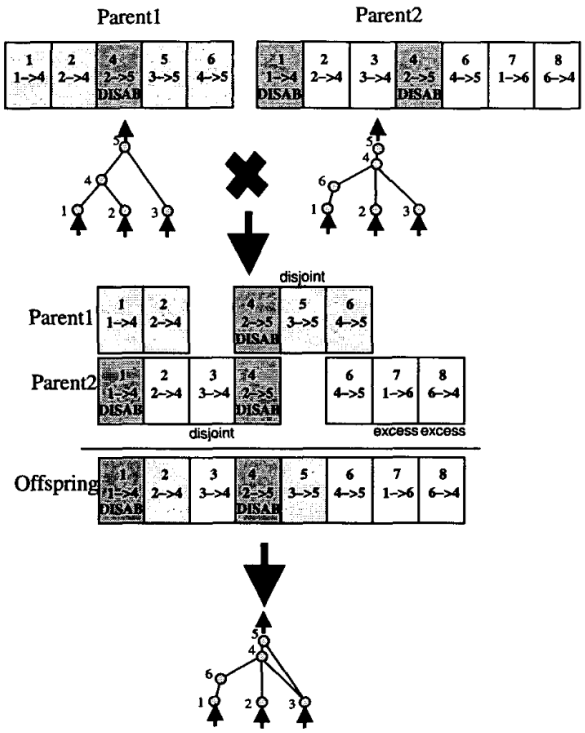
Figure 4.2a shows an example of genotype and phenotype in NEAT. The node genes define the nodes in the network, while the connections define all edges in the graph. They also contain an innovation number, which identify the historic ancestor of that gene, which will later be used in crossover.



(a) NEAT encoding of a network, containing node and connection genes. Every connection gene also contains an 'innovation number', that is later used in crossover.



(b) NEAT mutate, adding a connection or node. When a node is added (node 6 here), NEAT makes sure the total weight from node 5 to 3 remains unchanged. Finally, NEAT can mutate by removing connections.



(c) NEAT crossover, connections are aligned based on their 'innovation number', referring to their historic ancestor.

Figure 4.2: Neat evolutionary strategy.

Figure 4.2b shows a mutation in NEAT. NEAT can either mutate an existing connection, or add a new one. An existing connection can be mutated similar to described above with, for instance, Gaussian noise. NEAT can also add a connection between two existing nodes, initializing the edge with random

weight. Finally, it can add a node in between two previously connected nodes, making sure to keep the total weight between the two previous nodes the same (node 3 and 5 in the example).

As a final building block in NEAT, the crossover strategy takes care of something referred to as 'competing conventions'. In nature, strings of DNA/genomes are not just randomly aligned, they are sorted based on property before creating offspring. Based on this principle, it aligns the genomes based on innovation number, i.e., their historical marking. This strategy creates far more sensible networks, as compared to randomly crossing over genes, and is a great contribution in NEAT.

4.2.2. Novelty Seeking

As mentioned earlier, the greatest asset of neuroevolution is its ability to explore the solution space in great depth. However, even within neuroevolution exploration may be improved. As diversity drives innovation within the solution space, and many biologists agree it has been key in the development of human intelligence, it may help to improve neuroevolution performance too.

Early works focused on enabling diversity in the parameter space, i.e., diversity in the genetic space. The problem with this approach is that two distant networks may behave very similarly. In [63], the authors attempted to quantify different behavior between genomens, instead of its difference in the genetic space.

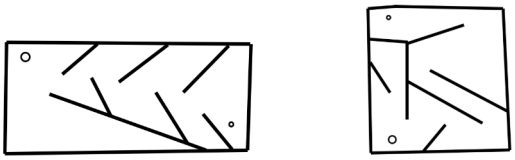


Figure 4.3: Two mazes tested in [63], left: 'medium' map, right 'hard' map. The hard map could only be solved by seeking novelty instead of a fitness function. The larger circle is the starting position, while the smaller dot is the target position.

The maps shown in Figure 4.3 were used to compare a novelty-driven approach to a fitness-based approach. In the fitness-based approach, the fitness function is computed based on the final distance to the goal, defined by $f = b_f - d_g$. Here fitness function f is equal to a bias b_f, subtracted by the distance from the robot's final position to the goal.

When seeking novelty, on the other hand, sparseness in the behavior space is sought after. The general formulation for this sparseness ρ at a point x is given by Equation 4.11.

$$\rho(x) = \frac{1}{k}\sum_{i=0}^{k} \text{dist}\,(x, \mu_i) \tag{4.11}$$

Hereby $\rho(x)$ is the average behavioral distance of x, as compared to its k nearest neighbors. μ_i is the ith-nearest neighbor.

A relevant question now is how to describe novelty in a specific application? In the case of the mazes of Figure 4.3, $\text{dist}\,(x, \mu_i)$ is defined as the euclidean distance between x and μ_i's positions at the end of their runs. In this application, the end position vector (x, y) is taken as a proxy for the behavior of the agent.

It might seems strange that a method solely seeking novelty outperforms a fitness-based strategy. The authors explain the superior performance by the concept of deceptive gradients. Even though neuroevolution does not necessarily use gradients to optimize its parameters (i.e., backpropagation), it still has to deal with local optima. Especially in harder tasks, many local optima exist that are incapable of solving the task. For instance, continuously flying `forward` may yield superior performance, as compared to random actions. Finding a (more complex) strategy that can outperform the `forward` network may not be easy, risking an entire `forward` population.

The stepping stones towards the global optimal solution may be very weak solutions, which may be the reason why novelty seeking neuroevolution can lead to high-performance solutions.

4.2.3. Indirect Encoding

The DNA-based genetic code in humans contains about 30,000 genes (or 3 billion base pairs). While still a lot, it is far less even compared to just the 100 trillion connections in our brain. Somehow our DNA

is a very efficient encoding of information, storing a tremendous amount of data in only about 30,000 genes. Each gene or pair of genes often represents a certain property, like colour of the eyes or hair. The same holds for the brain, with genes representing certain personality traits.

Taking this concept to neuroevolution, it may be non-optimal to quantify each connection and its properties directly in the genome. Instead, we may prefer *indirect encoding* of the network, dramatically reducing the solution space of the problem. A popular indirect encoding method uses compositional pattern-producing networks (CPPNs) [122]. CPPNs are a variation of ANNs, only now a wider variety of functions can be used, such as sinusoidal functions and many more complex activation functions. Usually they at least have a coordinate input (x, y), and sometimes parameters like d, the distance to the center of an image. Constructing images is one of the key applications of CPPNs, but they are very useful in neuroevolution too.

In an algorithm referred to as HyperNEAT [123], the CPPN generates a pattern of weights for the network that actually performs the task. In other words, we use a CPPN network to encode the weights of a higher dimensional neural network, as a function of the position of the edge in the network. We are encoding connectivity as a function of geometry, inspired from nature, where an embryo is constructed in a certain orientation using chemical gradients, making sure all parts grow in the correct position.

HyperNEAT has been successful by enhancing LSTM networks in natural language processing and other tasks.

4.2.4. Meta Learning

Meta learning, also learning to learn, can further improve model performance and allow for design of hybrid training strategies. Seeking comparison in nature, we can identify different 'loops' of optimization. The mating process in animals can be considered an evolutionary process, while inter-life learning is more like a gradient-based learning method, based on experiences.

Inspired by this example from nature, researchers at Google have developed a hybrid solution for image classification [99]. Their AmoebaNet was trained through conventional supervised learning, while the topography of the network was evolved, optimizing parameters such as number of nodes and type of layers. Before AmoebaNet, these were design choices made by humans, which explains the superior behavior by AmoebaNet. Another advantage of a hybrid like this is that it allows for massive parallelization, as each specific topography (or gene) can be trained on a separate GPU. Brute forcing the topography would have lead to impractical computational effort.

Another exciting application of meta learning is tackling catastrophic forgetting, a well-known phenomena in deep learning. While most animals have great talent for remembering and comparing memories, neural networks tend to 'forget' previous experiences. When learning new experiences or labels, previous experiences are typically forgotten, i.e., its performance is deteriorated. Neuromodulation offers an outer loop that turns on plasticity only for a subset of the neurons, with the aim of limiting impact on previous experiences. This means different tasks are primarily carried out in different parts of the network, similar to in a human brain. While it has been attempted with an evolutionary process, it has had limited success due to the high dimensions of the optimization problem.

4.2.5. Neuroevolution Success

Taking a step back, comparing the gradient-free neuroevolution and gradient-based reinforcement learning algorithms, it seems like disagreement exists about the future of reinforcement learning. Advocates of gradient-based reinforcement learning believe neuroevolution is unsuitable for the high-dimension solution space in the often vision-based RL, while neuroevolution specialists highlight the greater exploration of the solution space and massive parallelization.

By experiments in the Atari environments [88], neuroevolution has proven to be competitive with, or outperform gradient-based methods. The authors demonstrated that the algorithms are not only competitive in performance, they also train faster thanks to better parallelization.

4.3. Multi-Agent

We have now established a broad overview of RL algorithms in both gradient-based and gradient-free areas. For deployment in a multi-agent setup, another whole class of algorithms exist. In fact, thanks to the recent surge of interest in multi-agent reinforcement learning, an enormous body of work exist. Here, we provide a brief and selective overview of the different methods and refer the reader to more

in-dept reviews on MA-RL [87, 95, 143].

4.3.1. Multi-Agent Framework

To identify the framework within which we can deploy our MARL solution, we first define the environment as a Markov game [70]. A Markov game (Figure 4.4), is defined by a set of N agents, all receiving a rewards based on the set of actions by all agents. An interesting challenge in this configuration is that the environment is now non-stationary. In a single-agent environment, a Markov environment is assumed to be stationary. This changes in a multi-agent scenario, as the change in reward is no longer uniquely caused by the agent's own actions. It has to learn to perform optimally in a group.

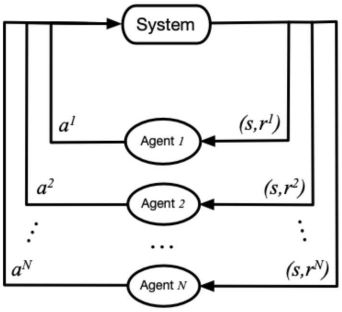


Figure 4.4: Markov game [70] definition.

Given a Markov environment, different configurations exist in reward shaping:

- Cooperative Setting: the reward of each agent is equal, i.e., $R^1 = R^2 = .. = R^N$. The agents co-operate to maximize reward. A slight variation is to optimize for a team-average reward, allowing for different rewards between agents, but with the final objective to maximize the final average reward.
- Competitive Setting: agents compete to receive highest reward, often in a zero-sum scheme, i.e., $\sum_{i \in \mathcal{N}} R^i(s, a, s') = 0$.
- Mixed Setting: each agent is self-interested, may have different rewards. This also includes teams of cooperative agents, that compete with other teams.

All categories contain a gigantic amount of contributions, so we focus on the most relevant category for this work: the cooperative setting.

4.3.2. Gradient-Based MARL

IQL

IQL [129], also independent Q-Learning, might be the most simple approach to MARL. Each agent is treated as an independent Q-Learning agent, making the other agents a part of the environment, attempting to optimize a joined reward. This usually doesn't work very well with DNNs, which led to Distributed Q-learning [61]. In Distributed Q-learning, all agents have a full observation of the system, only lacking the other agents' actions.

Fully Observable Critic

The non-stationary environment in MARL is one of its greatest challenges. It can be advantageous for each agent to only have access to its own state, when communication is unavailable for instance. In a fully observable critic algorithm, the actor only sees its own states and actions, while the critic approximates the value function based on the states of the entire environment and all agents. A well-known example of fully observable critic is MADDPG [78].

Value Function Factorization

Central RL approaches, with one policy controlling all agents, have yielded unsatisfying results. The main issue is that many times some agents get 'lazy' and do not contribute to the reward. Value function factorization may improve performance, assigning each agent a distinct part of the joint reward. It can

be hard to determine how much every agent contributed to the join reward function, numerous methods were proposed. They can be divided into two classes, being difference rewards [2] and potential-based reward shaping [94].

In difference rewards, the contribution by agent i can be described by $\hat{r}_i = r - r_{-i}$, where r is the global received reward and r_{-i} the contributions by all agents expect i. In potential-based reward shaping, r is replaced by $r + \Phi(s') - \Phi(s)$, where $\Phi(s)$ describes the user-defined desirability for an agent to be in a state s.

So, difference rewards are fairly similar to team-averaged rewards with individual rewards, while potential-based methods evaluate a specific state. A large amount of (complex) value function factorization functions exist, which are described in depth in [95].

Consensus

In an all-to-all communication network, each agent has access to the state and actions of the other agents. A drawback of this configurations is that it greatly increases the volume of information, which may overwhelm the capacity of the policies. Even if we can use a network with greater capacity, training will be longer and more unstable. Hence, in consensus-based MARL agents only receive information from a certain number of neighbors, reducing the information flow. Agents optimize actions mostly on their individual optimal actions.

Learning to Communicate

In a consensus-based algorithm we hand-design the messages and their destination, which is likely suboptimal, it becomes a hyper parameter. When learning to communicate [53], the agents learn to compose a message, as well as their destination and if they want to send a message at all. Again, in many different algorithms, this approach has been shown to be successful.

4.3.3. Gradient free MARL

Gradient free neuroevolution MARL may have less recent work, but is certainly a promising strategy for challenging multi-agent problems.

Multi-Component ESP

Coevolution is when two or more individuals are evolved in the same environment through parallel evolutionary processes. In a cooperative setting, the joint reward is equally distributed between agents, with a separate decision vector and policy for each agent. Similar to the gradient-based deep-RL case, it is beneficial to coevolve a solution as compared to one larger and higher dimensional solution.

Enforced SubPopulations (ESP) takes coevolution a step further, evolving parts of a neural network in separate evolutionary processes. Multi-Component ESP extends ESP to evolve a team of agents, evolving each network as an ESP. The team's reward is shared among the entire team, which is, similar to in gradient-based RL, a challenge, as the environment is now non-stationary. ESP-based solutions have been demonstrated in the predator-prey environment in [87], comparing performance with different forms of communication. The authors showed communicating agents with a shared reward is the optimal configuration in the predator-prey problem.

Social Learning

Up to now each agent was trained independently based on its observations of the environment. An alternative strategy is to deploy a teacher-student model, where high-performance individuals teach lower performance individuals based on their observations and actions. During its lifetime, a students learns from a teacher, after which they are evaluated for the evolutionary outer loop.

All the teacher' actions are assumed to be correct, which is its main weakness. A superior individual may not always perform optimal actions, possibly teaching the student wrong actions. Additionally, a teacher-student model increases the risk of converging to a local optima.

Egalitarian Social Learning (ESL) [87] is an alternative to the student-teacher model, sub dividing the population in subcultures. Agents can only learn from other agents in the subculture, created at the start of each generation. Actions are only used if it satisfies the user-defined acceptability function that filters out poor actions.

4.4. Applications to GSL

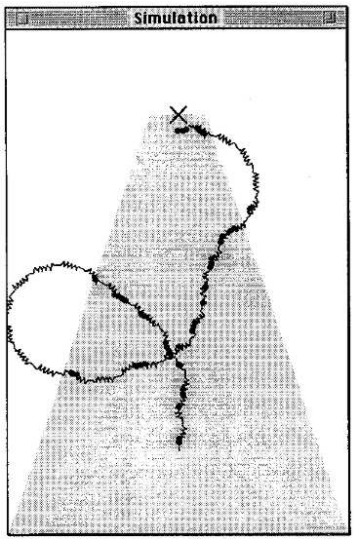
Although little RL-based GSL robots have been deployed in real-world experiments, plenty simulation experiments were established. First in [9, 59], later in [19, 46, 47, 133], RL-based GSL was explored.

[9] evolved continuous-time recurrent neural networks (CTRNNs) [8] for chemotaxis. CTRNNs are modeled as a system of ordinary differential equations, with neuron potential as the dependent variable, as presented in Equation 4.12.

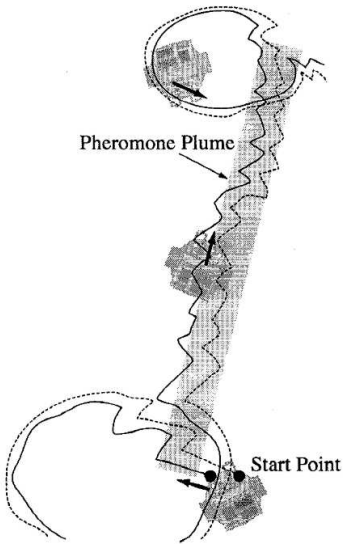
$$\tau_i \frac{dy_i}{dt} = -y_i + f_i\left(\beta_i + \sum_{j \in A_i} w_{ij} y_j\right) \tag{4.12}$$

Here τ_i is the time constant for neuron i, y_i the potential for neuron i, f_i the activation function for neuron i, β_i the bias of neuron i and finally, $\sum_{j \in A_i} w_{ij} y_j$ the sum of all inputs from neurons j into neuron i.

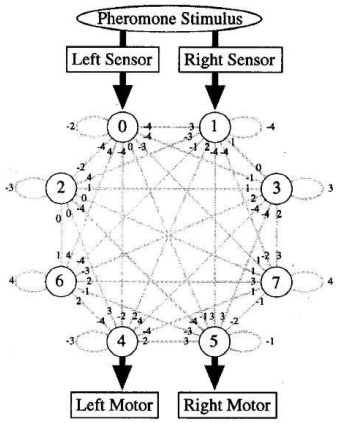
[59] synthesized the silkworm moth behavior by using living silkworm moth sensors and a recurrent neural network (RNN) for GSL. Using an evolved RNN with just eight neurons (Figure 4.5), the authors achieved behavior similar to that of a silkworm moth. The plume was modeled as a probability function within a trapezoid shape, giving higher probability of detection closer to the source.



(a) Simulation path for the evolved RNN. The simulator deploys a probability function within a trapezoid-shaped area, with higher detection probability close to the source.



(b) Trace of robot experiment tracing phermone gas, showing behavior similar to a silkworm moth.



(c) RNN trained by GA for GSL.

Figure 4.5: Synthesised moth from [59] performing GSL using two living sensors from a moth.

This work lacks thorough analysis of behavior and robustness, and the test environment is simplistic, small and without any obstacles. Even though it has shortcomings, it is the only RL-based GSL demonstration in real robot experiments.

[46, 47] are not typical robotics works, but rather contribute in the neuroscience and biology fields. [46] investigates learning multiple tasks, walking and chemotaxis, with a single RNN. [47] aims to model klinotaxis, a subset of chemotaxis, through a minimalistic neural network.

More recently in [19], an evolutionary robotics approach to GSL was demonstrated using CTRNNs. The authors show simulation results based on a non-holonomic vehicle with two wheels, which can only move forward. Four source features were designed, being composed of the wind vector, chemical reading and chemical gradient. The behavior of the RNN, even in turbulent conditions, is robust. The authors analyze the behavior and implement a finite state machine with similar behavior. The network is also capable of source declaration. The main shortcomings in this work are the lack of obstacles and robot experiments.

Finally, [133] implements a MARL approach for GSL. It implements a strategy-sharing algorithm in a large-scale advection-diffusion simulated plume environment. The authors demonstrate the effectiveness of group actions and strategy sharing, as it leads to higher fitness/rewards in their experiments.

Taking a step back, only [59] demonstrated robot experiments for a simple RL-based GSL strategy. The test area was incredibly small and analysis was incomplete, leaving us without a proof of concept for RL-based GSL in the real world. On a more positive note, RL-based strategies have proven to lead to high-performance solutions in simulation, which may eventually be transferred to the real world. It is uncertain what the impact of the sim2real gap will be on real-world performance.

5

Gas Modelling

This chapter provides a highly selective overview of the existing methods for gas modelling and the existing software tools. Ultimately we expose a need for randomized and complex GSL environments, with high-accuracy gas modelling. This chapter only contains selected topics in the interest of brevity. The reader is referred to an exhaustive review on dispersion modelling [40] if a deeper understanding in desired. We review Gaussian plume models (Section 5.1), advection and diffusion (Section 5.2), software (Section 5.3) and data sets (Section 5.4).

5.1. Gaussian Plume Models (GPM)

Gaussian Plume Models (GPM) [141] are most commonly used thanks to their small computational footprint and simplicity. The most simple form, which was first introduced in Section 2.2, models the plume as an expanding Gaussian curve in 3D space (Figure 2.5). With increasing distance downwind, the standard deviations σ_y, σ_z increase, creating a flatter Gaussian curve with the same surface area (i.e., the same amount of gas).

Though not applied in any GSL works, Gaussian-based approaches have been adopted for flow around objects [141]. A shortcoming is that the Gaussian-based approaches are designed for a specific shape of objects, such as a 3D cube. Another major shortcoming is the lack of stochastic behavior. In a bare implementation of a GPM, gradient-following may lead to an efficient solution to the GSL task.

GPM's are used for source term estimation (STE) on mobile robots, as it easily run on-edge. It has been successful mostly in large open spaces, where obstacles are not an issue and measurements can be averaged or filtered.

5.2. Advection and Diffusion

Advection and diffusion are in the core of the most high-performance gas modelling approaches. The intuitive idea is to generate a flow field for a given environment, and release particles, also filaments, to predict gas concentration.

The governing equations for advection were introduced in Section 2.2, Equation 2.2 described the basic idea. The position of a particle at the next timestep can be predicted by the sum of a deterministic term and a stochastic term. The deterministic term carries the particle along the wind vector, while the stochastic part adds a Gaussian distribution to model the stochastic behavior of gas. Diffusion means the particles are expanded over time. Note that a particle, also referred to as a filament, is not the same as a molecule, it contains many molecules.

5.2.1. Computational Fluid Dynamics (CFD)

Computational Fluid Dynamics (CFD) is the leading method to model flow around objects with high accuracy. The user defines boundary conditions of the environment, along with a solver of choice. Boundary conditions may be inlets,outlets or walls. By far the most well-known open source CFD software is OpenFOAM [48], accommodating a variety of solvers, boundary conditions and meshing tools. This section reviews CFD based on OpenFOAM examples.

Meshing

CFD sub-divides the environment into separate cells, to eventually iteratively solve the system of equations. Meshing an environment can be a remarkably time-intensive task, as it has great impact on the final solution. Human designers normally tweak the mesh based on their experience.

OpenFOAM's SnappyHexMesh tool takes trisurfaces as input, i.e., objects defined as a set of triangles such as CAD .stl files. It first intializes a mesh with even, relatively large, cells. It then iteratively adds smaller cells close to the object, as visible in Figure 5.2. The final step, after reaching the state of Figure 5.2, is to remove the cells that lie in the model.

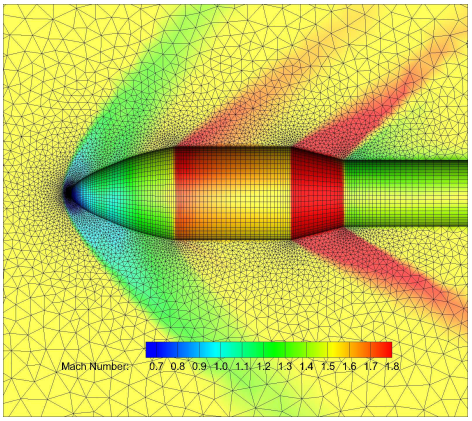


Figure 5.1: Rocket CFD by SU2 software [22]. The numerical cells get smaller for higher gradients of Mach Number.

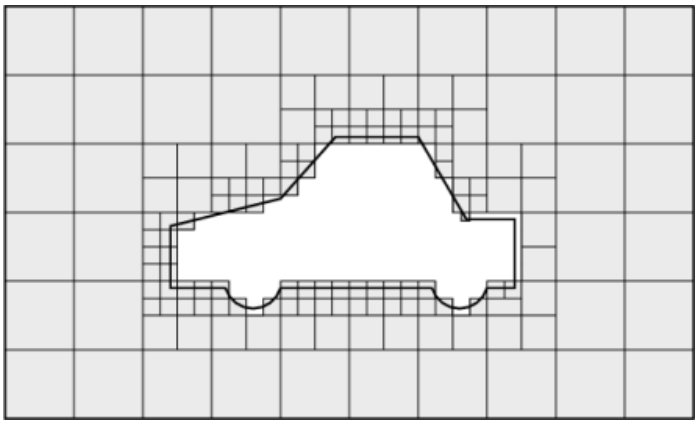


Figure 5.2: SnappyHexMesh in OpenFOAM [48]. SnappyHexMesh first divides the space into a square grid, and iteratively adds smaller cells towards the objects. In the final stage (not displayed here), the cells inside the object are removed.

The rationale is that closer to a surface higher gradients exist, hence smaller cells are required to maintain accuracy. Figure 5.1 shows an example of a super sonic rocket, with increasingly smaller cells with higher gradients, especially close to the tip of the rocket.

OpenFOAM's SnappyHexMesh is demonstrated here as an example, though many variants exist, generally with the same goal of more accuracy towards higher gradients.

Solvers

Again an enormous body of work exists on CFD solvers with different goals. We review an OpenFOAM solver referred to as PimpleFOAM as an example of a solver typically used to generate a flow field for gas dispersion modelling. PimpleFOAM is based on Reynolds-Averaged Simulation (RAS) k-epsilon model [62]. This RAS k-epsilon model was designed specifically to model turbulent flow, the condition that makes gas source localization so challenging.

The core of RAS k-epsilon is described by Equation 5.1.

$$\frac{D}{Dt}(\rho k) = \nabla \cdot (\rho D_k \nabla k) + P - \rho\epsilon \tag{5.1}$$

Here k is turbulent kinetic energy, D_k effective diffusivity for k, P turbulent kinetic energy production rate and ϵ the turbulent kinetic energy dissipation rate. The intuitive idea behind this model is that turbulent flow has turbulent kinetic energy, which is dissipated over time. Through another series of equations the velocity in a specific cell can be derived. The PimpleFOAM sets up the system of equations as a numerical problem, and solves not only for velocity, but also for other parameters such as pressure.

Boundary conditions

For the system of equations to be solvable, each surface will need to be assigned a boundary condition. An inlet boundary condition defines a velocity vector on a specific surface, or even a set of velocity vectors or function. The outlet may be modelled as a *zerogradient* boundary condition, forcing the change of velocity to have zero gradient in direction perpendicular to the surface. It may be intuitive to assume a forced wind vector on the outlet surface too, though this has two problems: 1) the size of the

outlet vector is not equal to the inlet wind speed, if the outlet has a different size (mass conservation) and 2) the outlet wind vector may differ over time. The second point can be disregarded if the flow is incompressible, which is usually true for velocities under Mach 0.3 (103 m/s), and the inlet velocity is constant.

Finally, the walls may be modeled as a 'noSlip' boundary condition, setting the entire wind vector at the patch to zero. Most intuitively, a wind vector can not travel inside a wall, as it is a solid structure. Additionally, air is not allowed to slip at the wall patch, meaning that the flow velocity parallel to the surface must be zero too. The layer of air just above the surface of a wall does move and has high turbulence, but at the surface air can't move.

5.3. Software

We review some of the software useful for developing GSL algorithms in simulation. Though these are useful, many researchers still decide to develop their own tools due to the lack of complete (open source) tools.

5.3.1. CFD

We already reviewed the OpenFOAM open source CFD toolbox, which is by far the most widely used CFD tool in academia. OpenFOAM acts as a framework, used by researchers to build their own new theory on top of, like CFD solutions for turbomachinery or rocketry.

By contrast, many commercial tools provide an out-of-the-box experience for specific applications. ANSYS Fluent [44] is an example of commercial software for CFD, providing various computational tool boxes from computational modelling. SimScale [112] is a commercial solution, adding a dedicated GUI and cloud compute to their OpenFOAM simulation back-end. CFD is a very computationally expensive task, making cloud compute a useful addition. More CFD-based solutions exist, but ANSYS and SimScale have previously been deployed in GSL contributions.

5.3.2. Pompy

PomPy is a NumPy-based implementation of a puff-based odour plume model, as described in [29]. Similar as to CFD-based gas dispersion models, Pompy models atmospheric dispersion in an open field by releasing 'puffs' in a wind field. Contrary to CFD-based models, Pompy computes the flow field by a series of mathematical equations and colored noise. It is unable to model flow around objects.

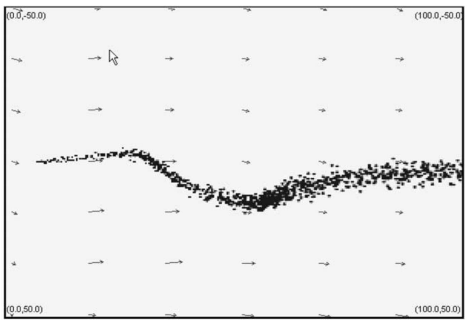


Figure 5.3: Pompy simulation results for a plume based on [29].

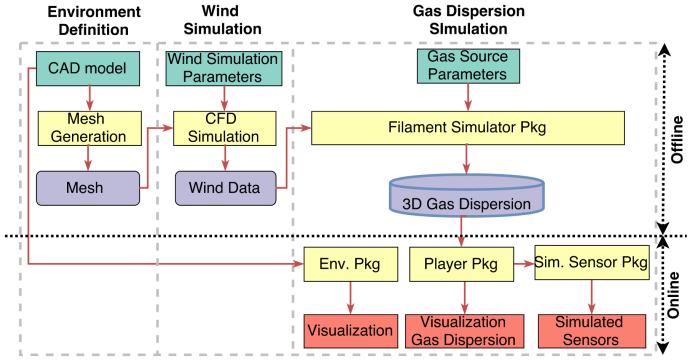


Figure 5.4: GADEN [90] pipeline for gas dispersion modelling. The 'Environment Definition' and 'Wind Simulation' stage are to be provided by the user, it is the input to the package.

5.3.3. GADEN

GADEN [90] is a ROS [120] package, capable of gas dispersion modelling and testing algorithms. Looking at Figure 5.4, the 'Environment Definition' and 'Wind Simulation' stage are to be provided by the user, it is the input to the package. GADEN's user manual recommends using a combination of CAD software and SimScale [112]. This approach is very labour intensive, as every testing environment would have to be hand-designed.

5.3.4. Software beneficial for GSL

While GADEN is a great tool, it misses the 'Environment Definition' and 'Wind Simulation' parts of the pipeline. Pompy can't deal with obstacles and data sets are too limited for exhaustive testing.

There is a need for a flexible and highly automated platform, capable of generating environments and dispersion modelling. As compared to GADEN, automation is required for:

- Environment generation to CAD model: OpenFOAM needs a CAD model of the flow volume in the environment.
- Meshing: OpenFOAM may start meshing based on the CAD file, though the instructions and configuration needs to be automated.
- Boundary conditions: based on the preferences of user, boundary conditions need to be set. It makes sense to add an inlet and outlet, as in most previous GSL work, though their position is unclear in a complex environment. Boundary condition allocation will have to be done dynamically.
- Solver: when running a computational method like CFD, conversion is not guaranteed. In fact, engineers using CFD often experience a lack of convergence in complex tasks. An automated CFD pipeline needs to be able to detect a diverging solver.

Such a pipeline is currently missing, but may make exhaustive testing and training of ML-based algorithms far more effortless.

5.4. Data Sets

Data sets of real data may be useful to compare and test algorithms, as compared to simulation data which may be over-simplified. Data can be recorded in two ways: 1) on a mobile robot, and 2) using a grid of sensors. Both have the issue that measurements are depend on the sensor used, which may impact readings substantially. When recording onboard a robot, more robot-specific phenomena appear, based on the dynamics of the robot (i.e., a MAV will record radically different values compared to a ground robot due to propellor inference).

In [119], a data set was created based on measurements of a robot with 'e-nose'. The authors then learned models based on those measurements. In [42], the authors tested their approach based on a data set provided in [102]. The data set, collected by COANDA R&D Corporation, was measured in a recirculating water channel.

In [35] compared their gas mapping strategy against others by their own self-created data set of their test environment. This work shows the main benefit of creating a data set: exhaustive testing and comparison. Even with the same the same gas source, conditions might vary drastically in between runs. The weakness of data sets is that they may not take into account the dynamic nature of gas, especially when recorded on a mobile robot.

6

Conclusion and Discussion

We have now arrived at the conclusion of this review. We have seen algorithms from different fields, including the biology, probability and learning communities. We have reviewed the algorithms based on their proven performance in different environments and discovered that testing was often limited. If the authors reviewed their strategies in simulation, the size and complexity of the environment varies significantly between contributions. The same holds for physical experiments, only now the robot and sensors differ too. Obstacles were almost never considered in the experiments and simulations.

For a more systematic approach to GSL, we need a software platform for complex environment generation and gas dispersion modelling. In that way, it will be possible to truly compare different strategies. This will also benefit reinforcement learning algorithms, which have yielded promising results in independent simulations, but have not yet undergone exhaustive testing in real experiments. A high-fidelity gas dispersion modelling platform is key in reducing the sim2real gap and hence the success of reinforcement learning in GSL. MARL solutions to GSL are also understudied. We have seen that single agents mimic creatures in some occasions, though it is yet to be discovered what behavior a multi-agent configuration will yield.

We also need more testing on nano scale, so far only one work has contributed experiments on a nano robot. Nano robots are uniquely equipped for GSL, as they are cheap and agile, though their form factor imposes unique challenges. Price is important, as a major use case for these vehicles is under-developed factories. Mostly factories in India and China with limited facilities have suffered from severe gas leaks and deaths over the past decade, making only very cheap robots an option. Nano robots, such as the Bitcraze Crazyflie, are orders of magnitude cheaper as compared larger scale ground robots. Hence, nano robots should be considered as serious options. More research is required to assess its feasibility, as currently only one study showed GSL on a nano drone, using a full map of the environment and external positioning.

As in most fields of robotics, a continued debate is required to figure out the exact use cases for gas-seeking robots. So far the success of robots in this problem has been very limited, especially fully autonomous systems. By considering opinions from both academia and industry, the field may develop in a way that is useful in emergency response.

Bibliography

[1] Julius Adler. The sensing of chemicals by bacteria. *Scientific American*, 234(4):40–47, 1976. ISSN 00368733, 19467087. URL http://www.jstor.org/stable/24950326.

[2] Adrian K. Agogino and Kagan Tumer. Unifying temporal and structural credit assignment problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '04, page 980–987, USA, 2004. IEEE Computer Society. ISBN 1581138644.

[3] Wolgang Alt. Biased random walk models for chemotaxis and related diffusion approximations. *Journal of Mathematical Biology*, 9(2):147–177, 1980. ISSN 14321416. doi: 10.1007/BF00275919.

[4] Edmund A. Arbas, Mark A. Willis, and Ryohei Kanzaki. Organization of goal-oriented locomotion: Pheromone-modulated flight behavior of moths. In *Proceedings of the Workshop on "Locomotion Control in Legged Invertebrates" on Biological Neural Networks in Invertebrate Neuroethology and Robotics*, page 159–198, USA, 1993. Academic Press Professional, Inc. ISBN 0120847280.

[5] Martin Asenov, Marius Rutkauskas, Derryck Reid, Kartic Subr, and Subramanian Ramamoorthy. Active localization of gas leaks using fluid simulation. *IEEE Robotics and Automation Letters*, 4 (2):1776–1783, 2019. ISSN 23773766. doi: 10.1109/LRA.2019.2895820.

[6] C. Barbieri, S. Cocco, and R. Monasson. On the trajectories and performance of infotaxis, an information-based greedy search algorithm. *EPL (Europhysics Letters)*, 94(2):20005, apr 2011. doi: 10.1209/0295-5075/94/20005. URL https://doi.org/10.1209%2F0295-5075%2F94%2F20005.

[7] Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients, 2018.

[8] Randall D. Beer. A dynamical systems perspective on agent-environment interaction. *Artificial Intelligence*, 72(1):173 – 215, 1995. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(94)00005-L. URL http://www.sciencedirect.com/science/article/pii/000437029400005L.

[9] Randall D. Beer and John C. Gallagher. *Evolving Dynamical Neural Networks for Adaptive Behavior*, volume 1. 1992. ISBN 1059712392001. doi: 10.1177/105971239200100105.

[10] Douglas A Berg, Howard C; Brown. Chemotaxis in E coli analysed by 3D tracking. *Nature*, 239: 500–504, 1972.

[11] M. R. Bonyadi and Z. Michalewicz. Stability analysis of the particle swarm optimization without stagnation assumption. *IEEE Transactions on Evolutionary Computation*, 20(5):814–819, 2016.

[12] J. R. Bourne, E. R. Pardyjak, and K. K. Leang. Coordinated bayesian-based bioinspired plume source term estimation and source seeking for mobile robots. *IEEE Transactions on Robotics*, 35(4):967–986, 2019.

[13] Valentino Braitenberg. *Vehicles: Experiments in synthetic psychology*. MIT press, 1986.

[14] Javier Burgués, Victor Hernández, Achim J. Lilienthal, and Santiago Marco. Smelling nano aerial vehicle for gas source localization and mapping. *Sensors (Switzerland)*, 19(3), 2019. ISSN 14248220. doi: 10.3390/s19030478.

[15] Mohammadreza Chamanbaz, David Mateo, Brandon M. Zoss, Grgur Tokí, Erik Wilhelm, Roland Bouffanais, and Dick K.P. Yue. Swarm-enabling technology for multi-robot systems. *Frontiers Robotics AI*, 4(APR):1–12, 2017. ISSN 22969144. doi: 10.3389/frobt.2017.00012.

[16] John Clegg. Signals in the animal world, by d. burkhardt, w. schleidt and h. altner. allen and unwin, 63s. *Oryx*, 9(5):372–373, 1968. doi: 10.1017/S0030605300007122.

[17] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995. ISSN 0885-6125. doi: 10.1023/A:1022627411411. URL https://doi.org/10.1023/A:1022627411411.

[18] X. Cui, C. T. Hardin, R. K. Ragade, and A. S. Elmaghraby. A swarm approach for emission sources localization. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 424–430, 2004.

[19] G. C.H.E. de Croon, L. M. O'Connor, C. Nicol, and D. Izzo. Evolutionary robotics approach to odor source localization. *Neurocomputing*, 121(December):481–497, 2013. ISSN 09252312. doi: 10.1016/j.neucom.2013.05.028.

[20] Jorge Miguel Dias Almeida Rodrigues Soares. Formation-based odour source localisation using distributed terrestrial and marine robotic systems. Technical report, EPFL, 2016.

[21] J. . Durrieu, J. . Thiran, and F. Kelly. Lower and upper bounds for approximation of the kullback-leibler divergence between gaussian mixture models. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4833–4836, 2012.

[22] Thomas D. Economon, Francisco Palacios, Sean R. Copeland, Trent W. Lukaczyk, and Juan J. Alonso. Su2: An open-source suite for multiphysics simulation and design. *AIAA Journal*, 54(3): 828–846, 2016. doi: 10.2514/1.J053813. URL https://doi.org/10.2514/1.J053813.

[23] Albert Einstein. Über die von der molekularkinetischen Theorie der Wärme gefordert Bewegung von in ruhenden Flüßigkeiten suspendierten Teilchen. (German) [On the molecular-kinetic theory of the movement by heat of particles suspended in liquids at rest]. 322(8):549–560, 1905. ISSN 0003-3804 (print), 1521-3889 (electronic). doi: https://doi.org/10.1002/andp.19053220806. URL http://www.nobelprize.org/nobel_prizes/physics/laureates/1926/;http://www.zbp.univie.ac.at/einstein/einstein2.pdf.

[24] R. Emery, F. Rahbar, A. Marjovi, and A. Martinoli. Adaptive lévy taxis for odor source localization in realistic environmental conditions. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3552–3559, 2017.

[25] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. *CoRR*, abs/1802.01561, 2018. URL http://arxiv.org/abs/1802.01561.

[26] J. Evans. *Optimization Algorithms for Networks and Graphs*. Routledge, 2017.

[27] J. E. Fackrell and A. G. Robins. Concentration fluctuations and fluxes in plumes from point sources in a turbulent boundary layer. *Journal of Fluid Mechanics*, 117:1–26, 1982. doi: 10.1017/S0022112082001499.

[28] Han Fan, Victor Manuel Hernandez Bennetts, Erik Schaffernicht, and Achim J. Lilienthal. Towards gas discrimination and mapping in emergency response scenarios using a mobile robot with an electronic nose. *Sensors (Basel, Switzerland)*, 19, 2019.

[29] J. A. Farrell, John Murlis, Xuezhu Long, Wei Li, and Ring T. Cardé. Filament-based atmospheric dispersion model to achieve short time-scale structure of odor plumes. *Environmental Fluid Mechanics*, 2:143–169, 2002.

[30] J. A. Farrell, Shuo Pang, and Wei Li. Plume mapping via hidden markov methods. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 33(6):850–863, 2003.

[31] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018. URL http://arxiv.org/abs/1802.09477.

[32] 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)*, pages 37–42, 2017.

[33] Mykola M. Glybovets and Nataliya M. Gulayeva. *Evolutionary Multimodal Optimization*, pages 137–181. Springer International Publishing, Cham, 2017. ISBN 978-3-319-68640-0. doi: 10.1007/978-3-319-68640-0_8. URL https://doi.org/10.1007/978-3-319-68640-0_8.

[34] FW Grasso, JH Dale, TR Consi, DC Mountain, and J Atema. Behavior of purely chemotactic robot lobster reveals different odor dispersal patterns in the jet region and the patch field of a turbulent plume. *The Biological bulletin*, 191(2):312—313, October 1996. ISSN 0006-3185. doi: 10.1086/bblv191n2p312. URL https://doi.org/10.1086/BBLv191n2p312.

[35] Javier G. Monroy, Jose Luis Blanco, and Javier Gonzalez-Jimenez. Time-variant gas distribution mapping with obstacle information. *Autonomous Robots*, 40(1):1–16, 2016. ISSN 15737527. doi: 10.1007/s10514-015-9437-0.

[36] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *CoRR*, abs/1812.05905, 2018. URL http://arxiv.org/abs/1812.05905.

[37] Hadi Hajieghrary, M. Ani Hsieh, and Ira B. Schwartz. Multi-agent search for source localization in a turbulent medium. *Physics Letters A*, 380(20):1698 – 1705, 2016. ISSN 0375-9601. doi: https://doi.org/10.1016/j.physleta.2016.03.013. URL http://www.sciencedirect.com/science/article/pii/S0375960116002425.

[38] A. T. Hayes, A. Martinoli, and R. M. Goodman. Distributed odor source localization. *IEEE Sensors Journal*, 2(3):260–271, 2002.

[39] X. He, J. R. Bourne, J. A. Steiner, C. Mortensen, K. C. Hoffman, C. J. Dudley, B. Rogers, D. M. Cropek, and K. K. Leang. Autonomous chemical-sensing aerial robot for urban/suburban environmental monitoring. *IEEE Systems Journal*, 13(3):3524–3535, 2019.

[40] Nick Holmes and Lidia Morawska. A review of dispersion modelling and its application to the dispersion of particles: An overview of different dispersion models available. *Atmospheric Environment*, 40:5902–5928, 09 2006. doi: 10.1016/j.atmosenv.2006.06.003.

[41] Wesley H. Huang and Kristopher R. Beevers. Topological map merging. *The International Journal of Robotics Research*, 24(8):601–613, 2005. doi: 10.1177/0278364905056348. URL https://doi.org/10.1177/0278364905056348.

[42] Michael Hutchinson, Hyondong Oh, and Wen Hua Chen. Entrotaxis as a strategy for autonomous search and source reconstruction in turbulent conditions. *Information Fusion*, 42(October 2017):179–189, 2018. ISSN 15662535. doi: 10.1016/j.inffus.2017.10.009. URL https://doi.org/10.1016/j.inffus.2017.10.009.

[43] Michael Hutchinson, Cunjia Liu, and Wen Hua Chen. Source term estimation of a hazardous airborne release using an unmanned aerial vehicle. *Journal of Field Robotics*, 36(4):797–817, 2019. ISSN 15564967. doi: 10.1002/rob.21844.

[44] ANSYS Inc. *ANSYS Fluent User's Guide*.

[45] H. Ishida, K. Suetsugu, T. Nakamoto, and T. Moriizumi. Study of autonomous mobile sensing system for localization of odor source using gas sensors and anemometric sensors. *Sensors and Actuators A: Physical*, 45(2):153 – 157, 1994. ISSN 0924-4247. doi: https://doi.org/10.1016/0924-4247(94)00829-9. URL http://www.sciencedirect.com/science/article/pii/0924424794008299.

[46] Eduardo J. Izquierdo and Thomas Buhrmann. Analysis of a dynamical recurrent neural network evolved for two qualitatively different tasks: Walking and chemotaxis. *Artificial Life XI: Proceedings of the 11th International Conference on the Simulation and Synthesis of Living Systems, ALIFE 2008*, pages 257–264, 2008.

[47] Eduardo J. Izquierdo and Shawn R. Lockery. Evolution and analysis of minimal neural circuits for klinotaxis in Caenorhabditis elegans. *Journal of Neuroscience*, 30(39):12908–12917, 2010. ISSN 15292401. doi: 10.1523/JNEUROSCI.2606-10.2010.

[48] Hrvoje Jasak. Openfoam: Open source cfd in research and industry. *International Journal of Naval Architecture and Ocean Engineering*, 1(2):89 – 94, 2009. ISSN 2092-6782. doi: https://doi.org/10.2478/IJNAOE-2013-0011. URL http://www.sciencedirect.com/science/article/pii/S2092678216303879.

[49] 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*, 2(2):37–51, 2007.

[50] Kamarulzaman Kamarudin, ali yeon md shakaff, Victor Bennetts, Syed Muhammad Mamduh Syed Zakaria, Ammar Zakaria, Retnam Visvanathan, Ahmad Shakaff Ali Yeon, and Latifah Kamarudin. Integrating slam and gas distribution mapping (slam-gdm) for real-time gas source localization. *Advanced Robotics*, 32:1–15, 09 2018. doi: 10.1080/01691864.2018.1516568.

[51] R. Kanzaki. Coordination of wing motion and walking suggests common control of zigzag motor program in a male silkworm moth. *Journal of Comparative Physiology - A Sensory, Neural, and Behavioral Physiology*, 182(3):267–276, 1998. ISSN 03407594. doi: 10.1007/s003590050177.

[52] Ryohei Kanzaki, Naoko Hata Sugi, and Tatsuaki Shibuya. Self-generated zigzag turning of bombyx mori males during pheromone-mediated upwind walking(physology). 1992.

[53] T. Kasai, H. Tenmoto, and A. Kamiya. Learning of communication codes in multi-agent reinforcement learning problem. In *2008 IEEE Conference on Soft Computing in Industrial Applications*, pages 1–6, 2008.

[54] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

[55] A. R. Khairuddin, M. S. Talib, and H. Haron. Review on simultaneous localization and mapping (slam). In *2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, pages 85–90, 2015.

[56] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.

[57] K Krishnanand, Varun Kompella, and Debasish Ghose. A network robot system for multiple odor source localization using glowworm swarm optimization algorithm. 07 2008.

[58] K. N. Krishnanand and D. Ghose. Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. *Swarm Intelligence*, 3(2):87–124, 2009. ISSN 19353812. doi: 10.1007/s11721-008-0021-5.

[59] Yoshihiko Kuwana, Isao Shimoyama, Yushi Sayama, and Hirofumi Miura. Synthesis of pheromone-oriented emergent behavior of a silkworm moth. *IEEE International Conference on Intelligent Robots and Systems*, 3:1722–1729, 1996. doi: 10.1109/iros.1996.569043.

[60] Yoshihiko Kuwana, Sumito Nagasawa, Isao Shimoyama, and Ryohei Kanzaki. Synthesis of the pheromone-oriented behaviour of silkworm moths by a mobile robot with moth antennae as pheromone sensors. this paper was presented at the fifth world congress on biosensors, berlin, germany, 3–5 june 1998.1. *Biosensors and Bioelectronics*, 14(2):195 – 202, 1999. ISSN 0956-5663. doi: https://doi.org/10.1016/S0956-5663(98)00106-7. URL http://www.sciencedirect.com/science/article/pii/S0956566398001067.

[61] Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*, pages 535–542. Morgan Kaufmann, 2000.

[62] B.E. Launder and D.B. Spalding. The numerical computation of turbulent flows. *Computer Methods in Applied Mechanics and Engineering*, 3(2):269 – 289, 1974. ISSN 0045-7825. doi: https://doi.org/10.1016/0045-7825(74)90029-2. URL http://www.sciencedirect.com/science/article/pii/0045782574900292.

[63] Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–222, 2011. ISSN 10636560. doi: 10.1162/EVCO_a_00025.

[64] Fei Li, Qing-Hao Meng, Shuang Bai, Ji-Gong Li, and Dorin Popescu. Probability-pso algorithm for multi-robot based odor source localization in ventilated indoor environments. In Caihua Xiong, Yongan Huang, Youlun Xiong, and Honghai Liu, editors, *Intelligent Robotics and Applications*, pages 1206–1215, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-88513-9.

[65] A. Lilienthal, H. Ulmer, H. Frohlich, A. Stutzle, F. Werner, and A. Zell. Gas source declaration with a mobile robot. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 2, pages 1430–1435 Vol.2, 2004.

[66] A. J. Lilienthal, M. Reggente, M. Trincavelli, J. L. Blanco, and J. Gonzalez. A statistical approach to gas distribution modelling with mobile robots - the kernel dm+v algorithm. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 570–576, 2009.

[67] Achim Lilienthal and Tom Duckett. Creating gas concentration gridmaps with a mobile robot. pages 118 – 123 vol.1, 10 2003. ISBN 0-7803-7860-1. doi: 10.1109/IROS.2003.1250615.

[68] Achim Lilienthal and Tom Duckett. Experimental analysis of gas-sensitive Braitenberg vehicles. *Advanced Robotics*, 18(8):817–834, 2004. ISSN 01691864. doi: 10.1163/1568553041738103.

[69] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

[70] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ICML'94, page 157–163, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc. ISBN 1558603352.

[71] Yang Liu, Prajit Ramachandran, Qiang Liu, and Jian Peng. Stein variational policy gradient. *CoRR*, abs/1704.02399, 2017. URL http://arxiv.org/abs/1704.02399.

[72] T. Lochmatter and A. Martinoli. Simulation experiments with bio-inspired algorithms for odor source localization in laminar wind flow. pages 437–443, 2008. doi: 10.1109/ICMLA.2008.128. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-60649101460&doi=10.1109%2fICMLA.2008.128&partnerID=40&md5=9651ec083d35e7b9a86de5d4ee4b847c. cited By 19.

[73] T. Lochmatter and A. Martinoli. Theoretical analysis of three bio-inspired plume tracking algorithms. pages 2661–2668, 2009. doi: 10.1109/ROBOT.2009.5152686. URL https://www.scopus.com/inward/record.uri?eid=2-s2.

0-70350383291&doi=10.1109%2fROBOT.2009.5152686&partnerID=40&md5=89e25e9353b9a741bc032de61ea66072. cited By 28.

[74] T. Lochmatter, X. Raemy, L. Matthey, S. Indra, and A. Martinoli. A comparison of casting and spiraling algorithms for odor source localization in laminar flow. In *2008 IEEE International Conference on Robotics and Automation*, pages 1138–1143, 2008.

[75] Thomas Lochmatter and Alcherio Martinoli. Tracking odor plumes in a laminar wind field with bio-inspired algorithms. In Oussama Khatib, Vijay Kumar, and George J. Pappas, editors, *Experimental Robotics*, pages 473–482, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-00196-3.

[76] Thomas Lochmatter and Alcherio Martinoli. *Understanding the Potential Impact of Multiple Robots in Odor Source Localization*, pages 239–250. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-00644-9. doi: 10.1007/978-3-642-00644-9_21. URL https://doi.org/10.1007/978-3-642-00644-9_21.

[77] Thomas Lochmatter, Ebru Aydın Göl, Iñaki Navarro, and Alcherio Martinoli. *A Plume Tracking Algorithm Based on Crosswind Formations*, pages 91–102. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-32723-0. doi: 10.1007/978-3-642-32723-0_7. URL https://doi.org/10.1007/978-3-642-32723-0_7.

[78] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275, 2017. URL http://arxiv.org/abs/1706.02275.

[79] C. Lytridis, E. E. Kadar, and G. S. Virk. A systematic approach to the problem of odour source localisation. *Autonomous Robots*, 20(3):261–276, 2006. ISSN 09295593. doi: 10.1007/s10514-006-7414-3.

[80] Chris Lytridis, Gurvinder Virk, Yann Rebour, and Endre Kadar. Odor-based navigational strategies for mobile agents. *Adaptive Behavior - ADAPT BEHAV*, 9:171–187, 09 2001. doi: 10.1177/10597123010093004.

[81] P. Maes, M. J. Mataric, J. Meyer, J. Pollack, and S. W. Wilson. *Some adaptive movements of animats with single symmetrical sensors*, pages 55–64. 1996.

[82] Agenor Mafra-neto and Ring T Carde. Orientation of Flying Moths. 369(May):142–144, 1994.

[83] A. Marjovi, J. Nunes, P. Sousa, R. Faria, and L. Marques. An olfactory-based robot swarm navigation method. In *2010 IEEE International Conference on Robotics and Automation*, pages 4958–4963, 2010.

[84] Ali Marjovi and Lino Marques. Multi-robot olfactory search in structured environments. *Robotics and Autonomous Systems*, 59(11):867 – 881, 2011. ISSN 0921-8890. doi: https://doi.org/10.1016/j.robot.2011.07.010. URL http://www.sciencedirect.com/science/article/pii/S0921889011001321.

[85] Ali Marjovi and Lino Marques. Optimal spatial formation of swarm robotic gas sensors in odor plume finding. *Autonomous Robots*, 35(2-3):93–109, 2013. doi: 10.1007/s10514-013-9336-1. URL http://infoscience.epfl.ch/record/197189.

[86] J-B Masson, M Bailly Bechet, and M Vergassola. Chasing information to search in random environments. *Journal of Physics A: Mathematical and Theoretical*, 42(43):434009, oct 2009. doi: 10.1088/1751-8113/42/43/434009. URL https://doi.org/10.1088%2F1751-8113%2F42%2F43%2F434009.

[87] Risto Miikkulainen, Eliana Feasley, Leif Johnson, Igor Karpov, Padmini Rajagopalan, Aditya Rawal, and Wesley Tansey. Multiagent learning through neuroevolution. In *IEEE World Congress on Computational Intelligence*, pages 24–46. Springer, 2012.

[88] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

[89] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[90] Javier Monroy, Victor Hernandez-Bennetts, Han Fan, Achim Lilienthal, and Javier Gonzalez-Jimenez. Gaden: A 3d gas dispersion simulator for mobile robot olfaction in realistic environments. *MDPI Sensors*, 17(7: 1479):1–16, 2017. ISSN 1424-8220. doi: 10.3390/s17071479. URL http://mapir.isa.uma.es/mapirwebsite/index.php/mapir-downloads/papers/259.

[91] Eduardo Martin Moraud and Dominique Martinez. Effectiveness and robustness of robot infotaxis for searching in dilute conditions. *Frontiers in Neurorobotics*, 4(MAR):1–8, 2010. ISSN 16625218. doi: 10.3389/fnbot.2010.00001.

[92] Patrick Neumann. *Gas Source Localization and Gas Distribution Mapping with a Micro-Drone*. PhD thesis, 06 2013.

[93] Patrick P. Neumann, Victor Hernandez Bennetts, Achim J. Lilienthal, Matthias Bartholmai, and Jochen H. Schiller. Gas source localization with a micro-drone using bio-inspired and particle filter-based algorithms. *Advanced Robotics*, 27(9):725–738, 2013. ISSN 01691864. doi: 10.1080/01691864.2013.779052.

[94] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, page 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1558606122.

[95] Afshin OroojlooyJadid and Davood Hajinezhad. A review of cooperative multi-agent deep reinforcement learning, 2019.

[96] S. Pang and J. A. Farrell. Chemical plume source localization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 36(5):1068–1080, 2006.

[97] Lauren Parker, James Butterworth, and Shan Luo. Fly safe: Aerial swarm robotics using force field particle swarm optimisation. *CoRR*, abs/1907.07647, 2019. URL http://arxiv.org/abs/1907.07647.

[98] Wilhelm Pfeffer. *Locomotorische Richtungsbewegungen durch chemische Reize:(Aus den" Untersuchungen aus dem botanischen Institut zu Tübingen Bd. I. Heft 3 p. 363-482).* W. Engelmann, 1884.

[99] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018. URL http://arxiv.org/abs/1802.01548.

[100] Pamela B. Reeder and Barry W. Ache. Chemotaxis in the florida spiny lobster, panulirus argus. *Animal Behaviour*, 28(3):831 – 839, 1980. ISSN 0003-3472. doi: https://doi.org/10.1016/S0003-3472(80)80143-6. URL http://www.sciencedirect.com/science/article/pii/S0003347280801436.

[101] Branko Risti, Daniel Angley, Bill Moran, and Jennifer Palmer. Autonomous multi-robot search for a hazardous source in a turbulent environment. *Sensors*, 17:918, 04 2017. doi: 10.3390/s17040918.

[102] Branko Ristic, Alex Skvortsov, and Ajith Gunatilaka. A study of cognitive strategies for an autonomous search. *Information Fusion*, 28:1 – 9, 2016. ISSN 1566-2535. doi: https://doi.org/10.1016/j.inffus.2015.06.008. URL http://www.sciencedirect.com/science/article/pii/S1566253515000597.

[103] G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical report, 1994.

[104] R. A. Russell. Chemical source location and the robomole project. 2003.

[105] R. Andrew Russell, Alireza Bab-Hadiashar, Rod L. Shepherd, and Gordon G. Wallace. A comparison of reactive robot chemotaxis algorithms. *Robotics and Autonomous Systems*, 45(2):83–97, 2003. ISSN 09218890. doi: 10.1016/S0921-8890(03)00120-9.

[106] Alvissalim Sakti, Mohamad Ivan Fanany, T Fukuda, Kosuke Sekiyama, and Wisnu Jatmiko. Robots implementation for odor source localization using pso algorithm. *WSEAS Transactions on Circuits and Systems*, 10, 04 2011. doi: 10.6084/m9.figshare.1609622.

[107] Michael Schmuker, Viktor Bahr, and Ramón Huerta. Exploiting plume structure to decode gas source distance using metal-oxide gas sensors. *Sensors and Actuators B: Chemical*, 235:636 – 646, 2016. ISSN 0925-4005. doi: https://doi.org/10.1016/j.snb.2016.05.098. URL http://www.sciencedirect.com/science/article/pii/S0925400516307833.

[108] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

[109] Mohit Sewak. Deep Reinforcement Learning. *Deep Reinforcement Learning*, (November), 2019. doi: 10.1007/978-981-13-8285-7.

[110] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1948.tb01338.x.

[111] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page I–387–I–395. JMLR.org, 2014.

[112] SimScale. URL https://www.simscale.com/.

[113] J. M. Soares, A. P. Aguiar, A. M. Pascoal, and A. Martinoli. A distributed formation-based odor source localization algorithm - design, implementation, and wind tunnel evaluation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1830–1836, 2015.

[114] J. M. Soares, A. P. Aguiar, A. M. Pascoal, and A. Martinoli. An algorithm for formation-based chemical plume tracing using robotic marine vehicles. In *OCEANS 2016 MTS/IEEE Monterey*, pages 1–8, 2016.

[115] Jorge M. Soares, A. Pedro Aguiar, António M. Pascoal, and Alcherio Martinoli. A graph-based formation algorithm for odor plume tracing. In Nak-Young Chong and Young-Jo Cho, editors, *Distributed Autonomous Robotic Systems*, pages 255–269, Tokyo, 2016. Springer Japan. ISBN 978-4-431-55879-8.

[116] Cheng Song, Yuyao He, Branko Ristic, and Xiaokang Lei. Collaborative infotaxis: Searching for a signal-emitting source based on particle filter and Gaussian fitting. *Robotics and Autonomous Systems*, 125:103414, 2020. ISSN 09218890. doi: 10.1016/j.robot.2019.103414. URL https://doi.org/10.1016/j.robot.2019.103414.

[117] K W Soo, K Kamarudin, V H Bennetts, R Visvanathan, A S Ali Yeon, S M Mamduh, and A Zakaria. SLAM-GDM implementation on robot operating system for gas source localization. *IOP Conference Series: Materials Science and Engineering*, 705:012018, dec 2019. doi: 10.1088/1757-899x/705/1/012018. URL https://doi.org/10.1088%2F1757-899x%2F705%2F1%2F012018.

[118] Diana Spears, Dimitri Zarzhitsky, and David Thayer. Multi-robot chemical plume tracing. In Lynne E. Parker, Frank E. Schneider, and Alan C. Schultz, editors, *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*, pages 211–221, Dordrecht, 2005. Springer Netherlands. ISBN 978-1-4020-3389-6.

[119] Cyrill Stachniss, Christian Plagemann, and Achim J. Lilienthal. Learning gas distribution models using sparse Gaussian process mixtures. *Autonomous Robots*, 26(2-3):187–202, 2009. ISSN 09295593. doi: 10.1007/s10514-009-9111-5.

[120] Stanford Artificial Intelligence Laboratory et al. Robotic operating system. URL https://www.ros.org.

[121] K. O. Stanley and R. Miikkulainen. Efficient evolution of neural network topologies. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 2, pages 1757–1762 vol.2, 2002.

[122] Kenneth Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, 8:131–162, 06 2007. doi: 10.1007/s10710-007-9028-8.

[123] Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artif. Life*, 15(2):185–212, April 2009. ISSN 1064-5462. doi: 10.1162/artl.2009.15.2.15202. URL https://doi.org/10.1162/artl.2009.15.2.15202.

[124] Kenneth O. Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019. ISSN 2522-5839. doi: 10.1038/s42256-018-0006-z. URL http://dx.doi.org/10.1038/s42256-018-0006-z.

[125] Jake A. Steiner, Joseph R. Bourne, Xiang He, Donald M. Cropek, and Kam K. Leang. Chemical-source localization using a swarm of decentralized unmanned aerial vehicles for urban/suburban environments. *ASME 2019 Dynamic Systems and Control Conference, DSCC 2019*, 3(February 2020), 2019. doi: 10.1115/DSCC2019-9099.

[126] Jake A. Steiner, Xiang He, Joseph R. Bourne, and Kam K. Leang. Open-sector rapid-reactive collision avoidance: Application in aerial robot navigation through outdoor unstructured environments. *Robotics Auton. Syst.*, 112:211–220, 2019.

[127] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998. URL http://www.cs.ualberta.ca/~sutton/book/the-book.html.

[128] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[129] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *In Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337. Morgan Kaufmann, 1993.

[130] Massimo Vergassola, Emmanuel Villermaux, and Boris I. Shraiman. 'Infotaxis' as a strategy for searching without gradients. *Nature*, 445(7126):406–409, 2007. ISSN 14764687. doi: 10.1038/nature05464.

[131] Nicole Voges, Antoine Chaffiol, Philippe Lucas, and Dominique Martinez. Reactive searching and infotaxis in odor source localization. *PLOS Computational Biology*, 10(10):1–13, 10 2014. doi: 10.1371/journal.pcbi.1003861. URL https://doi.org/10.1371/journal.pcbi.1003861.

[132] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *CoRR*, abs/1611.01224, 2016. URL http://arxiv.org/abs/1611.01224.

[133] Jian Long Wei, Qing Hao Meng, Ci Yan, Ming Zeng, and Wei Li. Multi-Robot gas-source localization based on reinforcement learning. *2012 IEEE International Conference on Robotics and Biomimetics, ROBIO 2012 - Conference Digest*, pages 1440–1445, 2012. doi: 10.1109/ROBIO.2012.6491171.

[134] Lilian Weng. Policy gradient algorithms. *lilianweng.github.io/lil-log*, 2018. URL https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html.

[135] Thomas Wiedemann, Christoph Manss, Dmitriy Shutin, Achim J. Lilienthal, Valentina Karolj, and Alberto Viseras. Probabilistic modeling of gas diffusion with partial differential equations for multi-robot exploration and gas source localization. *2017 European Conference on Mobile Robots, ECMR 2017*, 2017. doi: 10.1109/ECMR.2017.8098707.

[136] Peter C. Wilkinson. Chemotaxis. In Peter J. Delves, editor, *Encyclopedia of Immunology (Second Edition)*, pages 533 – 537. Elsevier, Oxford, second edition edition, 1998. ISBN 978-0-12-226765-9. doi: https://doi.org/10.1006/rwei.1999.0140. URL http://www.sciencedirect.com/science/article/pii/B012226765600150X.

[137] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

[138] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. URL http://arxiv.org/abs/1609.08144.

[139] Tristram D. Wyatt. Moth flights of fancy. *Nature*, 369(6476):98–99, 1994. ISSN 00280836. doi: 10.1038/369098a0.

[140] Xin xing Chen and Jian Huang. Odor source localization algorithms on mobile robots: A review and future outlook. *Robotics and Autonomous Systems*, 112(December):123–136, 2019. ISSN 09218890. doi: 10.1016/j.robot.2018.11.014.

[141] Paolo Zannetti. *Gaussian Models*, pages 141–183. Springer US, Boston, MA, 1990. ISBN 978-1-4757-4465-1. doi: 10.1007/978-1-4757-4465-1_7. URL https://doi.org/10.1007/978-1-4757-4465-1_7.

[142] Dimitri Zarzhitsky, Diana F. Spears, and William M. Spears. Swarms for chemical plume tracing. *Proceedings - 2005 IEEE Swarm Intelligence Symposium, SIS 2005*, 2005:257–264, 2005. doi: 10.1109/SIS.2005.1501629.

[143] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms, 2019.

[144] Y. Zhang, X. Ma, and Y. Miao. Localization of multiple odor sources using modified glowworm swarm optimization with collective robots. In *Proceedings of the 30th Chinese Control Conference*, pages 1899–1904, 2011.