GSL-Bench

High Fidelity Gas Source Localization Benchmarking

Hajo Erwich

TU Delft

GSL-Bench

High Fidelity Gas Source Localization Benchmarking

Thesis report

by

Hajo Erwich

to obtain the degree of Master of Science
at the Delft University of Technology
to be defended publicly on October 27, 2023 at 15:00

Thesis committee:	
Chair:	Prof. S. Hamaza
Supervisors:	Dr. G.C.H.E de Croon
	B. P. Duisterhof
External examiner:	Dr. R.T. Rajan
Place:	Faculty of Aerospace Engineering, Delft
Project Duration:	November, 2022 - October, 2023
Student number:	4543696

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

Faculty of Aerospace Engineering · Delft University of Technology

Preface

Almost a year after starting this final project of the aerospace engineering master, my time at the TU Delft has come to an end. This master's thesis is a continuation on previous work presented by my supervisors in 2020. They developed and tested a novel method for Gas Source Localization (GSL) named 'Sniffy Bug'. Unfortunately, the results from all GSL-related studies are difficult to compare due to the lack of a standardized testing method. This is how the need for GSL-Bench materialized.

This thesis consists of three parts. The first part contains a scientific article that was submitted to the 2024 IEEE International Conference on Robotics and Automation (ICRA). In the second part of this thesis, a literature study on GSL research is carried out to better formulate the requirements of a GSL benchmark. The third part presents the reader with additional results, concludes this work and gives recommendations.

This project has been a great learning experience for me, especially in the context of software engineering. The notable difference between code written at the start and end of my thesis emphasize this progress. Additionally, it has helped me learn to efficiently create software pipelines larger than I had ever done before.

I want to thank my supervisors Guido and Bart for the guidance they provided throughout this project. They helped me focus on the important aspects and kept me on track. On a personal note, I want to thank Tim, my friends and my family for their continued support. I could not have done it without them.

Hajo Henricus Erwich
Delft, October 2023

Contents

Nomenclature

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
C-PSO	Charged PSO
CAD	Computer Aided Design
CFD	Computational Fluid Dynamics
CTRNN	Continuous-time Recurrent Neural Network
DNN	Deep Neural Network
DR-PSO	Detection and Responding PSO
DRL	Deep Reinforcement Learning
EA	Evolutionary Algorithm
EMD	Earth Mover's Distance
FNN	Feedforward Neural Network
GDM	Gas Dispersion Modeling
GPS	Global Positioning System
GSL	Gas Source Localization
GSO	Glowworm Swarm Optimization
IMU	Inertial Measurement Unit
LiDAR	Light Detection And Ranging
LIF	Laser-Induced Fluorescence
LSTM	Long Short-Term Memory
M-GSO	Modified GSO
ML	Machine Learning
MOS	Metal Oxide Semiconductor
MOX	Metal Oxide
PID	Photo Ionization Detector
PISO	Pressure-Implicit with Splitting of Operators
PSO	Particle Swarm Optimization
PSO-WU	PSO with Wind Utilization
RL	Reinforcement Learning
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
ROS	Robot Operating System
S-PSO	Standard PSO
SDK	Software Development Kit
SR	Success Rate
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle

List of Figures

List of Tables

1 Introduction

Gas Source Localization (GSL) is a challenging subject matter within the robotics community. Unfortunately, no established simulated benchmark currently exists to quantify the relative performance of different approaches, see Table 1.1. From simulation to experimentation, a wide variety of GSL methods are put to the test on different platforms such as rovers [1], unmanned aerial vehicles [2] or even autonomous underwater vehicles [3]. Currently, emergency response to gas leaks involves humans localizing the source. Robots can alleviate risk in such critical and dangerous tasks. In practice these robots can be employed to locate faulty gas piping in outdoor or indoor environments. Or they may be used to identify potential underwater pipeline leaks. Additionally, they increase safety awareness by continuously monitoring a chemical plant or a construction site preemptively.

The difficult nature of GSL gives rise to different types of algorithms. Turbulent wind conditions in outdoor environments complicate the detection of a gas plume [4], while the stagnant air in indoor environments can create high local gas concentrations away from the actual source [5]. Bio inspired algorithms take inspiration from nature, in particular the silkworm moth [6] and the dung beetle [7]. Multi-agent methods like Particle Swarm Optimization (PSO) and its derivatives utilize multiple robots to locate the source [8, 9, 10]. In other research, probabilistic methods based on Bayesian inference [11, 12] or Hidden Markov Methods (HMM's) [13] try to model the possible gas distribution from the measurements, thereby estimating the source location. Finally, with the recent increase in popularity of Machine Learning (ML), methods such as reinforcement learning [14] and evolutionary algorithms [15] are used to approach GSL tasks. This 'exploration' of algorithms continues to advance the field of GSL.

Nevertheless, this progress can only be accelerated if these different methods can objectively be compared. This poses a substantial challenge. Without an established benchmark, researchers use their custom simulators for testing and evaluation. Simulations can differ in many ways thereby complicating objective comparison. The implementation of the physics, especially the gas plume model, largely influences the difficulty of the task. For example, an environment featuring a Gaussian plume model without obstacles [16, 17] is less challenging than an environment with obstacles and a turbulent plume [8, 18]. Moreover, a large portion of research evaluates methods in a relatively small, open environment [19, 20, 21], while some include obstacles. Even if environments are comparable, the performance metrics might still differ. Experimental repeatability can be complicated by the many environmental variables involved [1]. Some experiments hence take part in a wind tunnel [2], [6], [22].

A widely adopted simulated benchmark can alleviate these current limitations. With simulation, repeatability is straightforward. Additionally, simulation offers ground-truth information and runs faster than real-time with proper hardware and software implementation [23]. This is especially of interest to Machine Learning (ML) algorithms as they require training. In addition to potential time savings, simulation provides a save and cost-effective environment compared to its real-world counterpart [24].

The aim of this thesis is to develop a simulated benchmark that is accessible, features a small simulation to reality gap and provides useful performance metrics. The specifics of these requirements are investigated by means of a study of the literature. Concurrently, an automatic environment generation pipeline is developed capable of providing the benchmark with complex environments featuring turbulent wind flows and high fidelity gas dispersion data.

Table 1.1: Comparison of different gas dispersion environment simulators. Partly adapted from [25, 26]. *Automatic execution of multiple experiments and generation of performance metrics.

Research	Rendering Engine (Language/Framework)	Benchmarking*	Photo-realistic	Dimensionality	Obstacles	Wind Simulation	Gas Dispersion Simulation	Chemical Sensing	Wind Sensing
PlumeSim [27]	Player/Stage (C++)	✗	✗	2D	✗	Constant Wind Field	Gaussian/ Meandering	MOX	✗
Rahbar et al. [22]	Webots Odor Simulation (C/C++)	✗	✗	3D	✗	Constant Wind Field	Filament Model	Concentration + noise	✓
Awadalla et al. [18]	- (MATLAB)	✗	✗	3D	✓	CFD	CFD	Concentration	✗
GADEN [25]	Rviz (C++, ROS)	✗	✗	3D	✓	CFD	Filament Model	MOX, PID	✓
Sniffy Bug [5]	OpenGL (C++, Swarmulator [28])	✗	✗	2D	✓	CFD	Filament Model	Concentration	✗
Ojeda et al. [26]	Unity (C#, ROS)	✗	✓	3D	✓	CFD	Filament Model	MOX, PID	✓
GSL-Bench	NVIDIA® Isaac Sim (Python, Pegasus [29])	✓	✓	3D	✓	CFD	Filament Model	MOX, PID	✓

1.1. Research Objective

The research objective of this thesis is formulated as follows:

To objectively compare the performance of gas source localization methods by developing a high fidelity simulated benchmark capable of evaluating the performance of gas source localization methods in various environments.

1.2. Thesis Structure

The thesis is divided into three parts. Part I features the scientific article written for this thesis. This article was submitted to the 2024 IEEE International Conference on Robotics and Automation (ICRA). It concisely presents most of the work carried out for this thesis. It covers some of the related work and lays out the structure of the procedural environment generation pipeline and the simulated benchmark. Results of three different algorithms demonstrate the capabilities of GSL-Bench.

Part II contains the literature study carried out at the start of the thesis. This study explores different aspects of GSL research to better formulate the requirements of the benchmarking suite. Chapter 4 elaborates on the the different GSL methods presented by research. Subsequently, simulation aspects of GSL research such as simulation environments and gas dispersion simulation are presented in Chapter 5. Chapter 6 continues with an overview and discussion on current and future real-world GSL experimentation. Chapter 7 compares different performance metrics used throughout literature. Finally, the literature study is concluded in Chapter 8.

Part III presents the reader with additional results and closing remarks. Chapter 9 elaborates on the use and structure of the automated environment generation pipeline AutoGDM+. In Chapter 10 GSL-Bench is explained in more detail, namely the implemented algorithms and benchmarking methods. Additional results of GSL algorithms not featured in the paper are also presented. The thesis concludes in Chapter 11 and recommendations are given in Chapter 12.

Part I

Scientific Article

GSL-Bench: High Fidelity Gas Source Localization Benchmarking

Hajo H. Erwich[1], Bardienus P. Duisterhof[2], Guido C.H.E. de Croon[1]

***Abstract*— Gas Source Localization (GSL) is a challenging field of research within the robotics community, with high-stakes search-and-rescue applications. Existing methods vary widely and each has its own strengths and weaknesses. Comparisons of different methods are limited due to the lack of a broadly adopted and standardized testing methodology. Existing GSL evaluations vary in environment size, wind conditions, and gas simulation fidelity. They also lack photo-realistic rendering for the integration of obstacle avoidance. In this paper, we propose GSL-Bench, a benchmarking suite that can evaluate the performance of existing GSL algorithms. GSL-Bench features high-fidelity graphics and gas simulation, featuring NVIDIA's® Isaac Sim and OpenFOAM computational fluid dynamics software (CFD). Realism is further increased by simulating relevant gas and wind sensors. Scene generation is simplified with the introduction of AutoGDM+, capable of procedural environment generation, CFD and particle-based gas dispersion simulation. To illustrate GSL-Bench's capabilities, three algorithms are compared in six warehouse settings of increasing complexity: E. Coli, dung beetle and a random walker. Our results demonstrate GSL-Bench's ability to provide valuable insights into algorithm performance.**
Site: https://sites.google.com/view/gslbench/

I. INTRODUCTION

Gas Source Localisation (GSL) poses a challenging task for robotics. From simulation to experimentation a wide variety of GSL methods are deployed on different platforms such as rovers [1], [2], unmanned aerial vehicles (UAVs) [3], [4] or even autonomous underwater vehicles [5] (AUVs). Currently, emergency response to gas leaks involves humans localizing the source. Robots can alleviate risk in such critical and dangerous tasks. Additionally, they can monitor a site continuously in the case of large chemical plants.

The research field of GSL can only progress if different methods are to be objectively compared. This poses a substantial challenge as researchers develop and use their custom simulators for testing and evaluation. Simulations can differ in many ways thereby complicating objective comparison. The implementation of the physics, especially the gas dispersion model, largely influences the difficulty of the task. For example, an environment featuring a Gaussian plume model without obstacles [6], [7] is less challenging than an environment with obstacles and a turbulent plume [8], [9]. Moreover, a large portion of research evaluates methods in a relatively small, open environment [10], [11], [12], while only a few include obstacles. Even if environments are comparable, the performance metrics might still differ. Experimental repeatability can be complicated by the many environmental variables involved [1]. Some experiments therefore take part in a wind tunnel [4], [13], [14].

[1] Delft University of Technology [2] Carnegie Mellon University. Email: hajo_erwich@live.nl

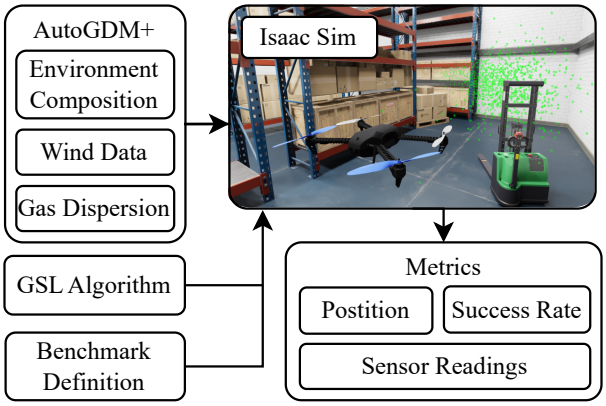


Fig. 1. System architecture of GSL-Bench. AutoGDM+ prepares the environments, wind and gas dispersion data. GSL-Bench performs the specified benchmark with the supplied GSL algorithm in Isaac Sim (gas filaments depicted as green dots), resulting in the metrics of interest.

Our contribution aims to alleviate these issues by making benchmarking more reliable, quicker, and simpler. In this paper, we present GSL-Bench, a benchmarking suite capable of benchmarking GSL algorithms, see Figure 1. Created with the NVIDIA's ® Isaac Sim [15] it features photorealistic environments and GPU accelerated rendering and physics. Gas and wind sensors are simulated using the same techniques presented by GADEN [16], a well-established gas dispersal simulator for ROS. Multiple simulation runs can be performed while metrics of interest are logged. New GSL algorithms are easily implemented in Python and do not have to account for obstacles due to the included obstacle avoidance module. Gas Dispersion Modelling (GDM) is simplified with the introduction of AutoGDM+ which is based on its predecessor AutoGDM [3]. AutoGDM+ automatically composes environments, generates (turbulent) wind data using OpenFOAM [17] and models the gas dispersion. Besides for benchmarking, the automatic creation of environments is also useful for reinforcement learning or evolutionary learning of GSL policies. This leads to the following contributions:

1) The first Gas Source Localization benchmarking suite featuring photo-realistic visuals with Isaac Sim and high fidelity wind and gas simulation by AutoGDM+. We release GSL-Bench open source for the benefit of the community.
2) A fully automated Gas Dispersion Modeling pipeline capable of generating multiple environments to use with GSL-Bench or any other simulation application.

3) We demonstrate both the capabilities of GSL-Bench and AutoGDM+ by generating environments and conducting simulated experiments using existing GSL algorithms.

II. RELATED WORK

GSL-Bench would not be how it is currently presented without existing work. We highlight these contributions and their respective strengths but also weaknesses that underscore the need for GSL-Bench. Existing benchmarks and experiments that inspired our work are further pointed out.

A. Simulation

Visual fidelity is crucial for robots that utilize vision for their obstacle avoidance. We want to support robot systems that rely on these techniques. However, the majority of GSL research is simulated with 2D simulators lacking any visuals such as Swarmulator [3], MATLAB [18], or a custom simulation [19]. Wiedemann et al. used Gazebo for their GSL simulation [20]. Gazebo is a 3D simulator that is commonly used in conjunction with Robot Operating System (ROS) but lacks visual fidelity [21].

The gas plume model is subject to various degrees of fidelity aswell. A Gaussian plume model depicts the time-averaged gas concentration as a function of location. It is straight forward in its implementation but assumes a non-turbulent flow and is incompatible with the presence of obstacles [6], [7]. Variations of the Gaussian plume model such as the meandering model was introduced by Cabrita et al. [22] for PlumeSim, a GSL simulator in Player/Stage.

A higher fidelity dispersion model is the 'advection-diffusion' model introduced by Farrel et al. [23]. In contrast to the Gaussian model it produces different gas concentrations over time for the same location and is compatible with turbulent wind conditions. Monroy et al. [16] used this gas model in GADEN: a 3D gas dispersion simulator for robot applications. As GADEN's simulations are only integrated with Rviz, the visualization toolbox of ROS, it lacks visual realism. It was therefore coupled with Unity, a game engine, by Ojeda et al. [24] to combine the realistic visuals with a high fidelity gas model.

A downside to Unity is the relative high complexity of its API [25]. On the other hand, Isaac Sim [15] is a designated robotics simulator featuring a Python API that can be considered less complex. Jacinto et al. [25] created a framework to primarily simulate aerial vehicles with Isaac Sim: Pegasus Simulator.

B. Benchmarking

The concept of GSL-Bench is partly inspired by Avoidbench: a vision-based obstacle avoidance benchmarking suite [26]. The structure of AvoidBench allows for swift implementation of one's own obstacle avoidance algorithm and produces meaningful metrics on the basis of some standardized tests.

Research by Voges et al. [27] serves as a good example of such tests in the context of GSL . Their work presents a thorough comparison of various algorithms by comparing success rate, trajectory length and deviation from the ideal trajectory. Similarly, research by Neumann et al. [28] compares three algorithms by simulation and experimentation with meaningful metrics. However, both the environment layouts and gas distributions used in these comparisons were of limited complexity.

III. METHOD

To encourage the adoption of GSL-Bench our focus is on accessibility, a small simulation-to-reality gap, and useful performance metrics. The inclusion of pre-generated environments and implementation of GSL algorithms with Python aid accessibility of the benchmark. Additionally, the combination of turbulent CFD, 3D filament gas modeling and high-fidelity visuals makes GSL-Bench one of the most realistic GSL simulations currently available. The benchmarking pipeline is split into two parts; the (optional) generation of environments with AutoGDM+ and the simulated benchmarking with GSL-Bench.

A. AutoGDM+

The workflow of AutoGDM+ consists of three components; 1) the layout generator, 2) CFD pipeline, and 3) gas dispersal modeling as shown by the grey boxes in Figure 2.

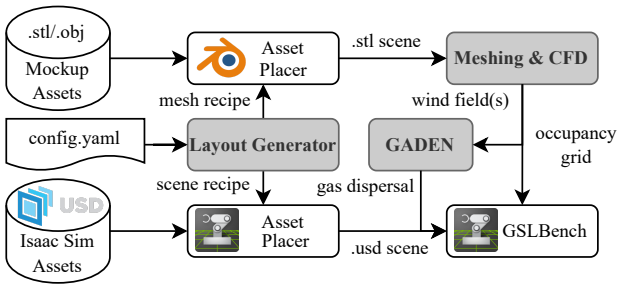


Fig. 2. Workflow of AutoGDM+. The respective asset placers create a scene from a generated layout. The .stl scene is used for meshing and CFD to generate wind fields and ultimately gas dispersion data. The .usd scene is used directly with GSL-Bench in Isaac Sim.

Environment generation starts with specifying the desired settings in a configuration file such as the type, size and the number of environments to be generated. This allows for the rapid generation of several variations of a given environment type, which is advantageous for Machine Learning (ML) applications [29], [30], [31]. Subsequently, 'recipes' are generated for the so-called 'asset placers'. One asset placer directly creates a scene for GSL-Bench in the Universal Scene Desription (.usd) format that is used throughout NVIDIA's ® Omniverse platform. Isaac Sim's high visual fidelity is partly thanks to the high-resolution assets it is bundled with. Because high-resolution assets complicate the meshing and CFD process, corresponding 'mockup assets' are used with the second asset placer implemented with Blender. Our manually created assets feature order of magnitudes less vertices but still capture the geometrical essence of the asset purely for CFD purposes. This results in a significant

performance increase of the CFD pipeline while retaining realistic wind data generation.

The meshing and CFD processes create a wind field according to the relevant settings in the configuration file. Most notable are the end time and time range. With larger indoor environments it takes time before the 'wind' has propagated fully through the environment. Therefore the user can directly select the wind fields that are of the most interest for the gas dispersal simulation. The mesh is created with openFOAM's `cartesianMesh` and CFD is performed with the `pimpleFoam` solver for transient, incompressible, turbulent flow [17]. The meshing and CFD processes are automatically multi-threaded for a significant speed increase.

The wind fields get passed to the gas dispersion modeling component of AutoGDM+ which is performed with GADEN [16]. GADEN makes use of the filament-based dispersion model [23] and has been validated with real-world experiments. The configuration file specifies the desired gas type, source location and intensity. Lastly, the output of GADEN is automatically post-processed for ease-of-use with GSL-Bench and other Python environments.

AutoGDM+ currently provides three indoor environment types: an empty, simple or complex warehouse. All three warehouse types make use of the same CFD setup featuring a velocity inlet and a pressure outlet in the near and far corner of the warehouse respectively, see Figure 3. The size of the inlet and outlet are configurable, as is the inlet velocity.

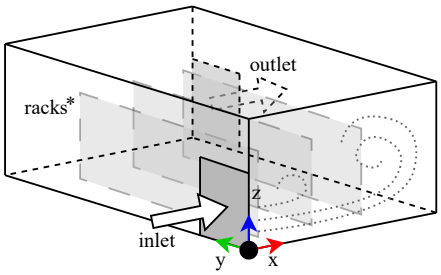


Fig. 3. CFD configuration of the warehouse environments. A velocity inlet and pressure outlet are situated near and far from the origin respectively. *Racks are only placed for the simple and complex warehouse types.

As the name implies, the empty warehouse features no interior. The simple warehouse includes rows of racks in the y-direction. The amount of rows, their height and length is adapted to the dimensions of the warehouse, inlet and outlet. Finally, the complex warehouse recipe orients the racks differently and places extra assets such as forklifts, pallets, piles of boxes etc. in the scene. These environments are well suited for this application because they can represent a warehouse storing various chemicals.

For this study six environments are generated. All environments are 15x15x8 meters in the XYZ direction respectively and are of the empty, simple and complex warehouse types. Parameters influencing the wind and gas simulation are identical across all environments (except for the source location), see Table I. A noteworthy parameter is the 'least percentage of filled racks'. There are nine different rack types of which two are empty and seven are filled. Although the layout generator chooses these types at random to introduce variation in the generated environments, some control is given back with this parameter. An overview of the different parameters is presented by Table II.

TABLE I

RELEVANT PARAMETERS IDENTICAL FOR ALL ENVIRONMENTS.

Parameter	Value
Inlet & Outlet Size (Y,Z) [m]	1.5, 2.4
Inlet Velocity (X,Y,Z) [m/s]	1.0, 0.0, 0.0
Least percentage of filled racks [-]	20%
CFD Cell Size [m]	0.2
CFD End Time [s]	5
Wind Steady-State	True
Gas Type	Ethanol
Gas Simulation End Time [s]	500
Gas Simulation Time Step [s]	0.2

TABLE II

DIFFERENT ENVIRONMENT PARAMETERS

Environment #	1	2	3	4	5	6
Type	empty		simple		complex	
Source Location (X,Y,Z) [m]	5,1,2	1,10,2	5,1,2	1,10,2	5,1,2	1,10,2

B. GSL-Bench

GSL-Bench makes use of the Pegasus Simulator framework [25] and features GSL related sensors, easy implementation of GSL algorithms and the necessary benchmarking features, see Figure 4. Although Pegasus Simulator can be used with a PX4 or ROS backend, GSL-Bench makes use of the Python backend and the included non-linear controller.

1) Sensors: Currently, one of the most common gas sensors are Metal Oxide (MOX) sensors due to their sensitivity to Volatile Organic Compounds (VOC's) and low cost [16]. GSL-Bench features different MOX sensor models. The downside of these sensors is their relatively slow response to changes in the instantaneous gas concentration. This behavior must therefore be accurately represented to keep the simulation to reality gap small. This is achieved with a low pass filter on the instant gas concentration with separate, calibrated time constants for rise and decay phases from the GADEN simulator [16]. For our experiments we use the simulated sensor model of the TGS2600.

In addition to the gas sensors, a wind sensor (anemometer) is implemented. It reads the generated wind fields and provides the wind heading and speed with the addition of noise. The heading is set to be in a range of $[-\pi, \pi]$, positive to the right and 0 pointing to the north. Because the Pegasus Simulator framework uses the 'east, north, up' (ENU) convention, north is in the positive y-direction of the inertial simulation frame. If desired, the raw velocity vectors of the wind field can also be provided.

2) Waypoint Logic & Obstacle Avoidance: Because the focus of the benchmark is GSL algorithm performance it

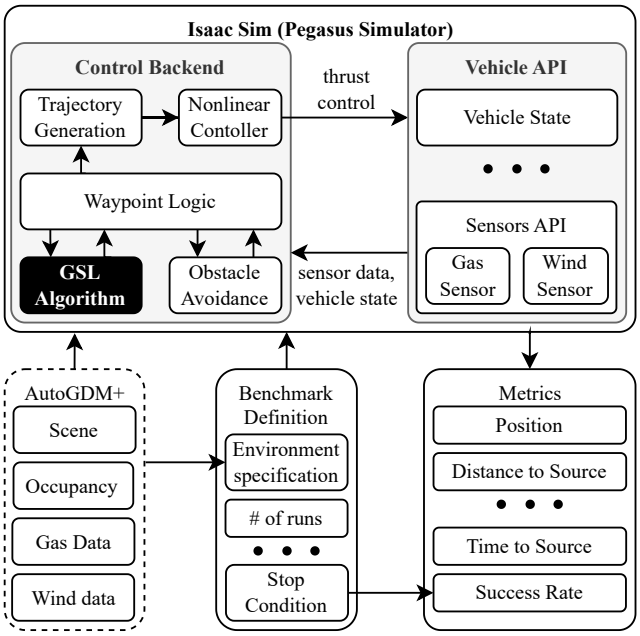


Fig. 4. System architecture of GSL-Bench. For Isaac Sim the Pegasus Simulator framework is used with the Python Control backend [25]. The generation of environments with AutoGDM+ is optional due to the availability of pre-generated environments. Only the GSL Algorithm (highlighted in black) is supplied by the user.

comes equipped with waypoint logic and rudimentary obstacle avoidance techniques. In its current implementation, a multirotor first takes off to the desired starting/search height. Then, the GSL algorithm provides a first goal waypoint. With the occupancy grid provided by AutoGDM+ this waypoint is checked for obstacles. If the path to the goal waypoint is clear, a trajectory is generated for the nonlinear controller to follow and the cycle repeats after a specified hold time. This hold time can be convenient for the gas sensor to respond to the new location and is set to 2 seconds for all algorithms. If the goal waypoint is in an obstacle, a new waypoint just outside of the obstacle is provided. If the goal waypoint is behind an obstacle however, a path is generated using the A^* algorithm which can consist of multiple waypoints called a 'mission', see Figure 5. GSL is resumed once the mission is completed. Lastly, please note that users can make their own real-time 'on-board' obstacle avoidance with the help of the sensors available in Isaac Sim.

C. GSL Algorithms

GSL-Bench currently features three algorithms; E. Coli, dung beetle and random walker. The E. Coli algorithm is included because it is a rudimentary bio-inspired algorithm. It is therefore featured in a fair share of research as a 'common denominator' [3], [27]. The dung beetle algorithm (also known as zig-zag) is considered the next a step with bio-inspired algorithms. It not only uses a chemical sensor but also makes use of the wind direction. Although these algorithms are quite elementary, they are well suited to demonstrate the capabilities of GSL-Bench.

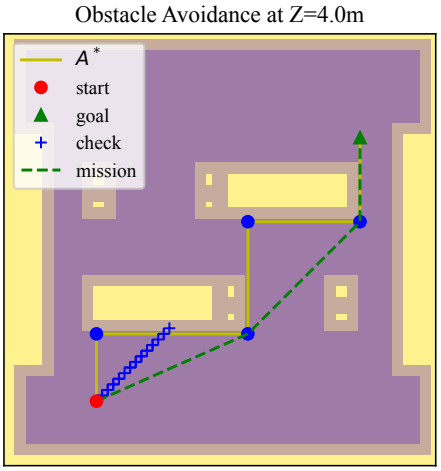


Fig. 5. Obstacle avoidance built into GSL-Bench. If the check runs into an obstacle, a mission is generated with an A^* algorithm.

1) E. Coli: E. Coli compares its current and previous gas readings. If the current reading shows improvement the agent surges with its previous heading. Otherwise the agent chooses a random heading for movement. By altering the surge distance the behavior of the algorithm can be more conservative or exploratory [32]. For these experiments the surge distance is set to 1 meter.

2) Dung Beetle: The dung beetle algorithm is analogous to the one featured in the work of Russel et al. [33]. Before any chemical detection, the agent moves in a direction 90 degrees to the left of the wind vector. Once gas is detected, the agent initiates zig-zagging ± 60 degrees to the wind vector, switching direction when less gas is detected.

3) Random Walker: The random walker is implemented as a variation of the E. Coli algorithm. In contrast to the E. Coli, it does not sense any gas concentration and therefore never surges with a repeated heading. This algorithm is implemented to provide a baseline performance indicator.

D. Metrics & Stop Conditions

A variety of metrics is recorded by GSL-Bench to evaluate the performance of algorithms in multiple ways. Currently, six metrics are implemented: success rate, ground tracks, distance to source, time to source and gas sensor readings. The definition of success is specified with the stop condition. For example, it can activate when the GSL algorithm declares the location of the source. In our case, the stop condition is set to end the simulation after 300 seconds (of simulated time) or when the agent is closer than 1.0 meter to the source. If the the stop condition was triggered due to its proximity to the source, the run is considered a success. Each experiment is repeated for 10 runs to determine the success rate. For every environment, experiments are conducted from a grid of nine starting positions, resulting in a total of 1620 simulated runs.

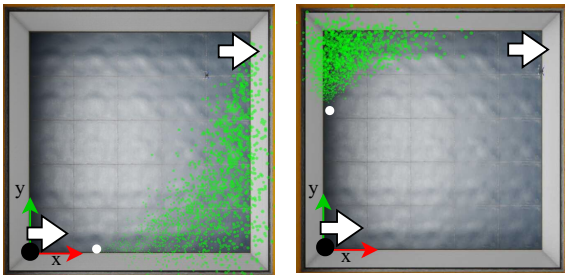


Fig. 6. Top-down view of environment 1 and 2 on the left and right respectively. The white arrows and dot indicate the in-/outlet, and source location. The green dots depict the gas filaments.

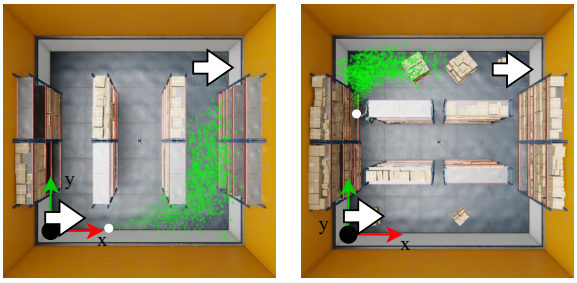


Fig. 7. Top-down view of environment 3 and 6 on the left and right respectively. The white arrows and dot indicate the in-/outlet, and source location. The green dots depict the gas filaments.

IV. RESULTS

The results of this study are twofold: (1) the generated environments and (2) algorithms performance. The generated environments show that minor changes in environment parameters can lead to a diverse set of challenges. We highlight some algorithms performance to showcase the various metrics that GSL-Bench offers.

A. Generated Environments

The generated environments present a diverse set of challenges. Each environment generation took approximately 90 seconds on a laptop with an 8-core AMD Ryzen 7 5800H CPU. A top-down view of environment 1 and 2 are shown by Figure 6. Intuitively, the gas dispersion is more pronounced when the source location (depicted by the white dot) is in front of the inlet. With slower wind speeds in the upper left corner of the environment, the gas plume is less spread out for the same duration of simulation time.

The simple warehouse type introduces storage racks in the scene as shown left in Figure 7. The size of the environment combined with the specified aisle dimensions determine the amount of rows, their length and height. Like in a real warehouse, the aisles are situated in line with the warehouse doors for easy transportation of goods (the doors are not visually modeled). From the top-down view the gas seems noticeably less dispersed due to the obstruction of the racks compared to the empty warehouse type. However, the racks introduce more vertical dispersion due to more pronounced upward wind flows they cause. This can increase the environment difficulty for algorithms that work only in the XY-plane because it decreases the plume cross section at the search height.

The complex warehouse type introduces assets such as piles and a forklift as shown right in Figure 7. The middle two rows of racks are rotated by 90 degrees resulting in tighter passages within the environment. The presence of the extra assets introduce even more turbulence into the scene.

B. Algorithm Performance Evaluation

We showcase various metrics with the data generated as described by Section III. Common plots including ground tracks or distance to source over time and more detailed plots such as success rates per location are shown.

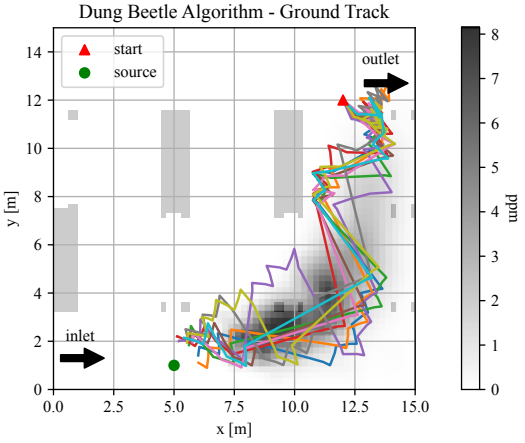


Fig. 8. Position traces of 10 Dung beetle runs in environment 3 from the upper right location. Obstacles are depicted as gray blocks. At first, the agents are stuck at the outlet but start zigzagging once some gas is detected.

1) Ground Tracks and Distance to Source: Ground track plots give an intuitive indication of an algorithm's behavior. A good example is shown by Figure 8. The algorithm seems to struggle at first, but when the gas plume reaches closer near the outlet the zigzagging is initiated. Depending on the run, the zigzagging starts at around 30 to 50 seconds as depicted by Figure 9. The distance to source plot illustrates an algorithm's time efficiency properly.

2) Success Rate: Success rate is ideal for benchmarking because of its clear, quantitative nature. We make a few observations from the success rate per environment in Figure 10. The performance of the dung beetle algorithm is heavily dependent on the source location. This is attributed to its initial 90 degree move to the left of the wind direction. Generally speaking, increased environment complexity decreases the success rate for all algorithms as expected.

Figure 11 offers additional insight into the success rates of environment 3. The success rate of E. Coli and the random walker seem correlated with the proximity of their starting location to the source. The E. Coli algorithm's underperformance can be attributed to the absence of a plume near the actual source location at the specified search height. This shows the importance of simulating gas plumes in 3D for the realistic evaluation of GSL algorithms. We

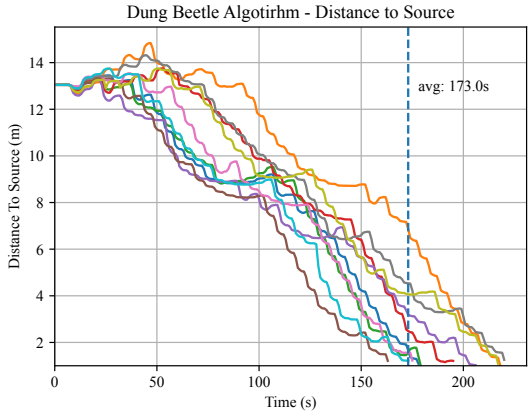


Fig. 9. Distance to source over time of 10 Dung beetle runs in environment 003 from the upper right location, highlighting the average time to the source.

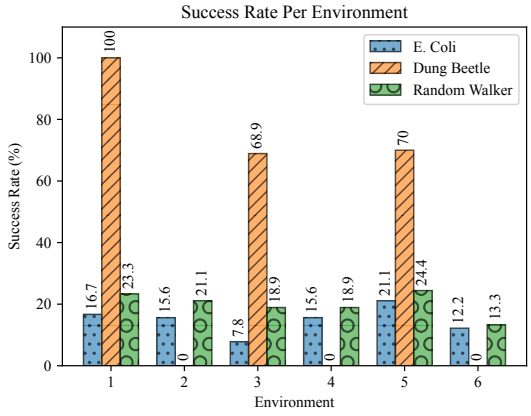


Fig. 10. Overall success rates per environment. The random walker outperforms the E. Coli algorithm in every environment, while the success of the dung beetle algorithm is heavily dependent on the source location.

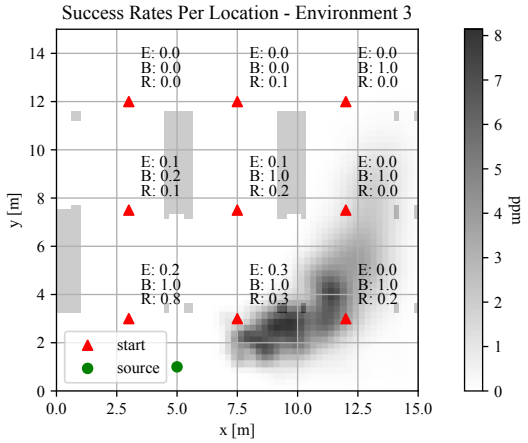


Fig. 11. Success rates for E. Coli (E), dung beetle (B) and random walker (R) for 10 runs per location in environment 3. Obstacles are depicted as gray blocks

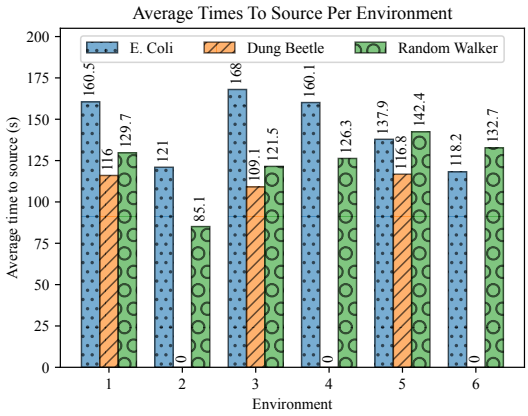


Fig. 12. Average time to source per environment. 0 indicates the absence of a measurement because no runs located the source.

also observe how obstacles hinder the dung beetle algorithm knowing that it first tries to move towards the lower right corner of the environment due to the wind.

3) Average Time to source: The average time to source is another quantitative measure of performance shown by Figure 12. Runs that did not find the source are disregarded. Therefore, this metric must be combined with success rate to provide a meaningful indication. The performance of the dung beetle algorithm is consistent when it is able to find the source. E. Coli and random walker are less uniform as can be expected due to the randomness in their logic.

V. CONCLUSION

We have introduced GSL-Bench, a Gas Source Localization benchmarking suite. We focus on accessibility and simulation fidelity to promote widespread adoption. Due to the built-in obstacle avoidance, waypoint logic and sensors one can directly concentrate their efforts on only the GSL algorithm. However, the built-in modules can easily be omitted to keep development flexible. High fidelity of the simulation is ensured with proper visuals and the introduction of AutoGDM+: a fully automated environment generation pipeline relying on validated software such as GADEN and OpenFOAM. We generate six warehouse environments with increasing complexity and benchmark three algorithms. The results demonstrate GSL-Bench's ability to thoroughly test and visualize algorithm performance. Moreover, the performance of algorithms align with other experiments from literature, further validating our pipeline.

In future work, we continue further development of GSL-Bench and AutoGDM+. We plan to introduce extra algorithms. These include multi-agent algorithms and probabilistic methods such as Particle Swarm Optimization (PSO) and infotaxis respectively. Additionally, outdoor environment types and more complex indoor environments will be added to AutoGDM+. Ultimately, we aspire for GSL-Bench to effectively support the research community and spark fresh interest within the gas source localization field. Our website will provide our most recent findings and host a leaderboard showcasing results from diverse standardized tests.

REFERENCES

[1] B. L. Villarreal, G. Olague, and J. L. Gordillo, "Synthesis of odor tracking algorithms with genetic programming," *Neurocomputing*, vol. 175, pp. 1019–1032, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2015.09.108

[2] Q. H. Meng, W. X. Yang, Y. Wang, and M. Zeng, "Collective odor source estimation and search in time-variant airflow environments using mobile robots," *Sensors*, vol. 11, no. 11, pp. 10 415–10 443, 2011.

[3] B. P. Duisterhof, S. Li, J. Burgues, V. J. Reddi, and G. C. De Croon, "Sniffy Bug: A Fully Autonomous Swarm of Gas-Seeking Nano Quadcopters in Cluttered Environments," *IEEE International Conference on Intelligent Robots and Systems*, pp. 9099–9106, 2021.

[4] C. Ercolani and A. Martinoli, "3D odor source localization using a micro aerial vehicle: System design and performance evaluation," *IEEE International Conference on Intelligent Robots and Systems*, pp. 6194–6200, 2020.

[5] S. Pang and J. A. Farrell, "Chemical plume source localization," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 36, no. 5, pp. 1068–1080, 2006.

[6] C. Song, Y. He, B. Ristic, and X. Lei, "Collaborative infotaxis: Searching for a signal-emitting source based on particle filter and Gaussian fitting," *Robotics and Autonomous Systems*, vol. 125, p. 103414, 2020. [Online]. Available: https://doi.org/10.1016/j.robot.2019.103414

[7] K. Gaurav, A. Kumar, and R. Singh, "Single and multiple odor source localization using hybrid nature-inspired algorithm," *Sadhana - Academy Proceedings in Engineering Sciences*, vol. 45, no. 1, pp. 1–19, 2020. [Online]. Available: https://doi.org/10.1007/s12046-020-1318-3

[8] W. Jatmiko, K. Sekiyama, and T. Fukuda, "A Mobile Robots PSO-based for Odor Source Localization in Dynamic Advection-Diffusion Environment," *International Conference on Intelligent Robots and Systems*, pp. 4527–4532, 2006.

[9] M. Awadalla, T. F. Lu, Z. Tian, B. Dally, and Z. Liu, "3D framework combining CFD and MATLAB techniques for plume source localization research," *Building and Environment*, vol. 70, pp. 10–19, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.buildenv.2013.07.021

[10] A. Marjovi and L. Marques, "Optimal spatial formation of swarm robotic gas sensors in odor plume finding," *Auton Robot*, vol. 35, pp. 93–109, 2013.

[11] G. Ferri, M. V. Jakuba, A. Mondini, V. Mattoli, B. Mazzolai, D. R. Yoerger, and P. Dario, "Mapping multiple gas/odor sources in an uncontrolled indoor environment using a Bayesian occupancy grid mapping based method," *Robotics and Autonomous Systems*, vol. 59, no. 11, pp. 988–1000, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2011.06.007

[12] V. H. Bennetts, A. J. Lilienthal, P. P. Neumann, and M. Trincavelli, "Mobile robots for localizing gas emission sources on landfill sites: Is bio-inspiration the way to go?" *Frontiers in Neuroengineering*, vol. 4, no. JANUARY, pp. 1–12, 2012.

[13] H. Ishida, K. Hayashi, M. Takakusaki, T. Nakamoto, T. Moriizumi, and R. Kanzaki, "Odour-source localization system mimicking behaviour of silkworm moth," pp. 225–230, 1995.

[14] F. Rahbar, A. Marjovi, P. Kibleur, and A. Martinoli, "A 3-D bio-inspired odor source localization and its validation in realistic environmental conditions," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe, pp. 3983–3989, 2017.

[15] NVIDIA Corporation, "Isaac Sim," 2023. [Online]. Available: https://developer.nvidia.com/isaac-sim

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

[17] H. Jasak, "OpenFOAM: Open source CFD in research and industry," *International Journal of Naval Architecture and Ocean Engineering*, vol. 1, no. 2, pp. 89–94, 12 2009.

[18] U. Jain, R. Tiwari, and W. W. Godfrey, "Multiple odor source localization using diverse-PSO and group-based strategies in an unknown environment," *Journal of Computational Science*, vol. 34, pp. 33–47, 2019. [Online]. Available: https://doi.org/10.1016/j.jocs.2019.04.008

[19] 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*, pp. 37–51, 2007.

[20] T. Wiedemann, C. Vlaicu, J. Josifovski, and A. Viseras, "Robotic information gathering with reinforcement learning assisted by domain knowledge: An application to gas source localization," *IEEE Access*, vol. 9, pp. 13 159–13 172, 2021.

[21] S. Ivaldi, V. Padois, and F. Nori, "Tools for dynamics simulation of robots: a survey based on user feedback," pp. 1–15, 2014. [Online]. Available: http://arxiv.org/abs/1402.7050

[22] G. Cabrita, P. Sousa, and L. Marques, "PlumeSim-Player/Stage Plume Simulator," *ICRA Workshop on Networked and Mobile Robot Olfaction in Natural, Dynamic Environments*, 2010. [Online]. Available: papers://15a17785-8386-4a79-b6ae-1c6e2d0ed658/Paper/p5861

[23] J. A. Farrell, J. Murlis, X. Long, W. Li, and R. T. Cardé, "Filament-based atmospheric dispersion model to achieve short time-scale structure of odor plumes," *Environmental Fluid Mechanics*, vol. 2, no. 1-2, pp. 143–169, 2002.

[24] P. Ojeda, J. Monroy, and J. Gonzalez-Jimenez, "A simulation framework for the integration of artificial olfaction into multi-sensor mobile robots," *Sensors*, vol. 21, no. 6, pp. 1–13, 2021.

[25] M. Jacinto, J. Pinto, J. Patrikar, J. Keller, R. Cunha, S. Scherer, and A. Pascoal, "Pegasus Simulator: An Isaac Sim Framework for Multiple Aerial Vehicles Simulation," 7 2023. [Online]. Available: http://arxiv.org/abs/2307.05263

[26] H. Yu, G. De Croon, and C. De Wagter, "AvoidBench: A high-fidelity vision-based obstacle avoidance benchmarking suite for multi-rotors," 2023.

[27] N. Voges, A. Chaffiol, P. Lucas, and D. Martinez, "Reactive Searching and Infotaxis in Odor Source Localization," *PLoS Computational Biology*, vol. 10, no. 10, 2014.

[28] P. P. Neumann, V. Hernandez Bennetts, A. J. Lilienthal, M. Bartholmai, and J. H. Schiller, "Gas source localization with a micro-drone using bio-inspired and particle filter-based algorithms," *Advanced Robotics*, vol. 27, no. 9, pp. 725–738, 2013.

[29] G. C. de Croon, L. M. O'Connor, C. Nicol, and D. Izzo, "Evolutionary robotics approach to odor source localization," *Neurocomputing*, vol. 121, pp. 481–497, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2013.05.028

[30] H. Hu, S. Song, and C. L. Chen, "Plume Tracing via Model-Free Reinforcement Learning Method," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 8, pp. 2515–2527, 2019.

[31] Y. Zhao, B. Chen, X. H. Wang, Z. Zhu, Y. Wang, G. Cheng, R. Wang, R. Wang, M. He, and Y. Liu, "A deep reinforcement learning based searching method for source localization," *Information Sciences*, vol. 588, pp. 67–81, 2022. [Online]. Available: https://doi.org/10.1016/j.ins.2021.12.041

[32] T. F. Lu, "Indoor odour source localisation using robot: Initial location and surge distance matter?" *Robotics and Autonomous Systems*, vol. 61, no. 6, pp. 637–647, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2013.02.002

[33] R. A. Russell, A. Bab-Hadiashar, R. L. Shepherd, and G. G. Wallace, "A comparison of reactive robot chemotaxis algorithms," *Robotics and Autonomous Systems*, vol. 45, no. 2, pp. 83–97, 2003.

Part II

Literature Review

*This part has been assessed for the course AE4020 Literature Study.

3

Introduction

In robotics, Gas Source Localization (GSL) is a popular subject matter. From simulation to experimentation a wide variety of GSL methods are put to the test on different platforms such as rovers [1], unmanned aerial vehicles [2] or even autonomous underwater vehicles [3]. Currently, emergency response to gas leaks involves humans localizing the source. Robots can therefore alleviate risk in such critical and dangerous tasks. Additionally, they can monitor a site continuously in the case of large chemical plants for example.

Nevertheless, the research field of GSL can only progress if different methods can be objectively compared. This poses a substantial challenge. Without an established benchmark, researchers use their custom simulators for testing and evaluation. Simulations can differ in many ways thereby complicating objective comparison. The implementation of the physics, especially the gas plume model, largely influences the difficulty of the task. For example, an environment featuring a Gaussian plume model without obstacles is less challenging than an environment with obstacles and a turbulent plume. Moreover, a large portion of research evaluates methods in a relatively small, open environment, while some include obstacles. Even if environments are comparable the performance metrics might still differ. Furthermore, repeatability in real-world experimentation is complicated by the chaotic nature of gas flows. Not to mention the limited availability of real-world data due to the time constraints involved with experimentation.

A proper simulated benchmark can alleviate these current limitations. Therefore, the research objective of the thesis is as follows: ”To objectively compare the performance of gas source localization methods by developing a high fidelity simulated benchmark capable of evaluating the performance of gas source localization methods in various environments”. With simulation, repeatability is straightforward. Additionally, simulation offers ground-truth information and runs faster than real-time with proper hardware and software implementation [23]. This is especially of interest to Machine Learning (ML) algorithms as they require training. In addition to potential time savings, simulation provides a save and cost-effective environment compared to its real-world counterpart [24].

This literature study aims to answer the main question; what is the best way to implement a GSL benchmark, such that it is accessible, has an acceptable simulation-to-reality (sim2real) gap, and provides useful performance metrics? To accomplish this, the literature study contributes in three ways:

- Firstly, existing GSL methods are studied. Their working principle and possible simulation requirements are highlighted. Moreover, a gap in comparisons is identified further establishing the need for a GSL benchmark.
- Secondly, the current state of (GSL) simulation environments is surveyed. To make a simulator/benchmark accessible it is important to consider existing solutions, their popularity, advantages and disadvantages. Furthermore, current experimentation methods are examined to extend the usefulness of the benchmark into the real world.
- Lastly, the applicability of varying performance metrics is analyzed. What performance metrics are commonly used? Some GSL methods favor particular metrics, while others are sometimes inapplicable. The different metrics and their practicality will therefore be elaborated upon.

This literature study is set out as follows. Chapter 4 touches upon the different GSL methods presented in research, their characteristics, advantages and implications for a simulator. Additionally, existing method comparisons are studied. Then in Chapter 5 the current state of simulation environments is elaborated upon.

Chapter 6 discusses real-world experimentation, thereby gaining insight into what kind of benchmarks would translate well into the real world. Lastly, Chapter 7 compares different performance metrics used throughout the literature. Finally, the literature review concludes in Chapter 8.

4

Gas Source Localization Methods

Gas Source Localization (GSL) methods in the literature vary widely from simple bio-inspired algorithms requiring no memory to computationally intensive inference methods. Some informative review papers report on the wide variety of algorithms [30, 31, 32]. This chapter touches upon common types of methods in order to build an understanding of the different approaches and what that would mean for a potential simulator. The methods are divided into four categories: bio-inspired, multi-agent, probabilistic and machine learning methods and are covered in their respective section. Finally, method comparisons in the literature are reviewed and analyzed.

4.1. Bio-Inspired GSL Methods

The field of GSL research mainly started by looking at nature. Many early algorithms are therefore bio-inspired [33, 6, 7]. Although bio-inspired methods are one of the first algorithms to be used, they still prove to be relevant to this day, usually to compare them to a novel algorithm [34]. They characterize themselves by being simple, elegant and requiring little memory.

Bio-inspired algorithms can be divided into chemotaxis and chemotaxis-anemotaxis types of methods. Chemotaxis refers to the locomotion of an organism based on a chemical stimulus. In the case of GSL research, it means that a robot is able to sense the gas and act in order to find the odour source. With chemotaxis-anemotaxis, in addition to the gas information a robot can detect information about the wind direction and/or velocity.

Arguably the most elementary implementation of a bio-inspired algorithm is the E. Coli algorithm [7]. Derived from the behavior of the real bacteria E. Coli, it chooses between two distinct actions based on a chemical stimulus. If E. coli senses a lower chemical concentration after moving, it moves randomly to a new location. If the chemical concentration becomes higher however E. coli keeps moving in the same direction. Therefore, E. coli only has to memorize the chemical concentration at its previous location thus requiring little computational resources.

More involved is the so-called dung beetle algorithm, also known as the 'zig-zag' algorithm. Once the chemical is sensed the beetle keeps moving diagonally upwind. If the trial is lost the beetle turns around 90 degrees through the direction of the wind resulting in a zig-zag pattern, see Figure 4.1. Chen et al. experimented with and compared the effect of different/variable turning angles for the zig-zag algorithm in diffusive conditions [35].

Finally, there is the silkworm moth algorithm, also known as a 'cast-and-surge' method. This algorithm is derived from the actual behavior of the silkworm moth [36]. By a combination of upwind surges, side-to-side casting and spirals is the moth able to traverse towards an odor source, see Figure 4.1. Additionally, the algorithm is more elaborate due to the use of two chemical sensors left and right (antennae) compared to only one sensor used by the E. Coli and dung beetle algorithms.

4.2. Multi-Agent GSL Methods

Multi-agent approaches, or swarming is popular within GSL research. Although one could deploy multiple agents independently, the idea with swarming is that a multi-agent setup performs better than the sum of its parts. This philosophy is attributed to three key properties of swarms. Firstly, multi-agent setups are more

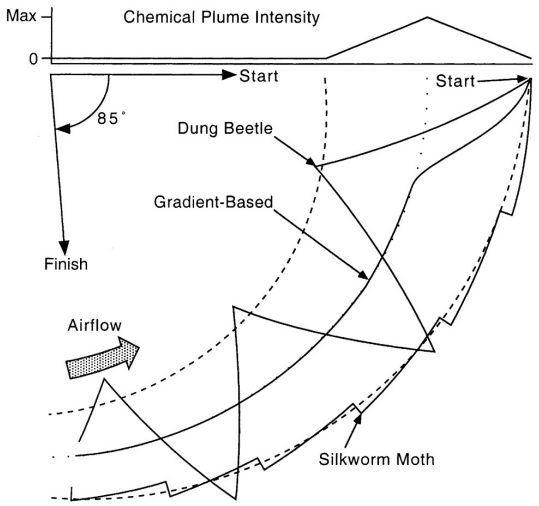


Figure 4.1: Simulation results of different bio-inspired algorithms by Russel et al. [7].

robust. If an agent were to break down it does not immediately result in a failed mission. Secondly, swarms provide flexibility as together they can perform a multitude of tasks an individual would be unable to. Lastly, multi-agent setups are scalable, making them suited for tasks of different scales [37]. This section first covers formation searching algorithms. Subsequently, particle swarm optimization is elaborated upon.

4.2.1. Formation Searching

Formation searching deploys multiple robots in a formation to increase the GSL performance. One of the earliest works in GSL by Genovese et al. [38] investigates the GSL performance of a robot formation, see Figure 4.2. The agents are programmed to exhibit different 'personalities' (sociable, lonely) based on their sensor input and that of the others around them. This swarming behavior automatically results in having the highest density of robots in the area of highest interest.

Another implementation relies on applying a virtual force to all robots so that they form a polygon formation, see Figure 4.3 [39]. Commanded by a virtual robot in the center of the formation, they move in the direction of the robot sensing the highest chemical concentration. Instead of using virtual forces, the formation can also be predefined such as a 'five-node square' or 'inverted V' formation [40]. In this case, the distances between the agents are determined dynamically. In both cases, however, the swarm essentially acts as one large robot with the advantage that the sensors have significant spacial separation.

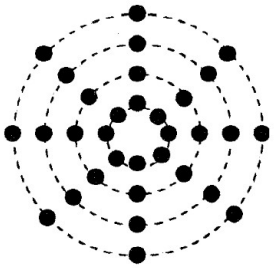


Figure 4.2: Start formation used in [38].

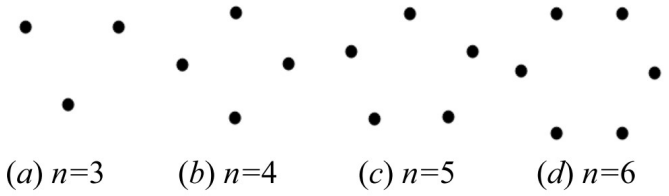


Figure 4.3: Polygon formation for different amount of agents [39].

4.2.2. Particle Swarm Optimisation

Particle Swarm Optimization (PSO) is a computational method which iteratively solves for an optimum solution. The algorithm makes use of multiple candidate solutions (particles) that traverse the solution space. PSO lends itself well to multi-agent OSL tasks because the particles can be represented by the

agents. Additionally, the implementation of Standard PSO (S-PSO) is relatively straightforward and requires no prior knowledge of the odor source. There are a few caveats to PSO however. Firstly, it requires some parameters to be set accordingly to perform satisfactorily. Moreover, PSO is not guaranteed to find a global optimum, and might 'get stuck' in a local optimum. Finally, PSO struggles with sparse and dynamic data prevalent in turbulent instead of diffusive conditions [41]. Additionally, S-PSO does not guarantee the proper localization of multiple gas sources at once.

Variations of PSO have been introduced to overcome these limitations. First, S-PSO is explained in more detail. Subsequently, some notable variants of PSO are elaborated upon such as Detection and Responding PSO (DR-PSO), Charged PSO (C-PSO) and PSO with Wind Utilization (PSO-WU). Note that many other variants of PSO incorporate probability estimation [10] or can locate multiple odor sources by the implementation of certain 'group behaviors' [42].

Standard PSO

S-PSO was introduced by Kennedy et al. in 1995 [43], and it is outlined by algorithm 1. Although the gradient of f is unknown S-PSO will try to find a point a for which $f(a) \leq f(b)$. The amount of particles in the swarm is denoted by S. Let $\mathbf{x}_i$, $\mathbf{v}_i$ and $\mathbf{p}_i$ be each particle's position, velocity and best-known position respectively. Let $\mathbf{g}$ be the swarm's best position. When the velocity of each particle is updated there are three important parameters to take into account. Increasing the weight (inertia) w will prevent the particles from erratically changing directions. ϕ_p and ϕ_g are called the cognitive coefficient and social coefficient respectively. They determine how much the particle will follow its own versus the swarm's best estimate. A visualization of S-PSO finding a plume is shown by Figure 4.4 (a).

Algorithm 1: Standard PSO

for *each particle* $i = 1, ..., S$ **do**
  $\mathbf{x}_i \sim U(\mathbf{b}_{lo}, \mathbf{b}_{up})$; `// initialize the particle's position`
  $\mathbf{p}_i \leftarrow \mathbf{x}_i$; `// initialize the particle's best known position`
  if $f(\mathbf{p}_i) < f(\mathbf{g})$ **then**
    $\mathbf{g} \leftarrow \mathbf{p}_i$; `// update the swarm's best known position`
  $\mathbf{v}_i \sim U(-|\mathbf{b}_{up} - \mathbf{b}_{lo}|, |\mathbf{b}_{up} - \mathbf{b}_{lo}|)$; `// initialize the particle's velocity`
while *a termination criterion is not met* **do**
  for *each particle* $i = 1, ..., S$ **do**
    for *each dimension* $d = 1, ..., n$ **do**
      $r_p, r_g \sim U(0, 1)$; `// pick random numbers`
      $\mathbf{v}_{i,d} \leftarrow w\mathbf{v}_{i,d} + \phi_p r_p(\mathbf{p}_{i,d} - \mathbf{x}_{i,d}) + \phi_g r_g(\mathbf{g}_d - \mathbf{x}_{i,d})$; `// update velocity`
      $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i$; `// update position`
      if $f(\mathbf{x}_i) < f(\mathbf{p}_i)$ **then**
        $\mathbf{p}_i \leftarrow \mathbf{x}_i$; `// update the particle's best known position`
        if $f(\mathbf{p}_i) < f(\mathbf{g})$ **then**
          $\mathbf{g} \leftarrow \mathbf{p}_i$; `// update the swarm's best known position`

Detection and Responding PSO

To make PSO more versatile in dynamic systems, Detection and Responding PSO (DR-PSO) is introduced [9, 8]. If $\mathbf{g}$, the swarm's best estimate is not changed for a set amount of iterations it may imply that there is another solution to be found. The swarm responds by randomly spreading its particles for an iteration in order to explore outside the previously found solution. Therefore this approach performs better in dynamic conditions compared to S-PSO, as shown by Figure 4.4 (b).

Charged PSO

Another variant of PSO is Charged PSO (C-PSO) [44, 8]. In addition to the 'neutral' particles used in S-PSO, 'charged' particles are introduced. The charged particles repel each other according to Coulomb's law, see Equation 4.1. Every charged particle has a core and perception radius, labeled r_{core} and r_{perc}. Within the core radius, particles are repelled by a constant strong force. Between the core and perception radius, the repellent force is a function of the distance between the particles. Outside of the perception radius, there is no repellent force. This force is calculated and summed for each particle and added to its velocity update, resulting in the behavior shown in Figure 4.4 (c).

$$\mathbf{a}_i = \begin{cases} \frac{Q_i Q_p (\mathbf{x}_i - \mathbf{x}_p)}{r_{core}^2 |\mathbf{x}_i - \mathbf{x}_p|} & |\mathbf{x}_i - \mathbf{x}_p| < r_{core} \\ \frac{Q_i Q_p (\mathbf{x}_i - \mathbf{x}_p)}{|\mathbf{x}_i - \mathbf{x}_p|^3} & r_{core} < |\mathbf{x}_i - \mathbf{x}_p| < r_{perc} \\ 0 & r_{perc} < |\mathbf{x}_i - \mathbf{x}_p| \end{cases} \tag{4.1}$$

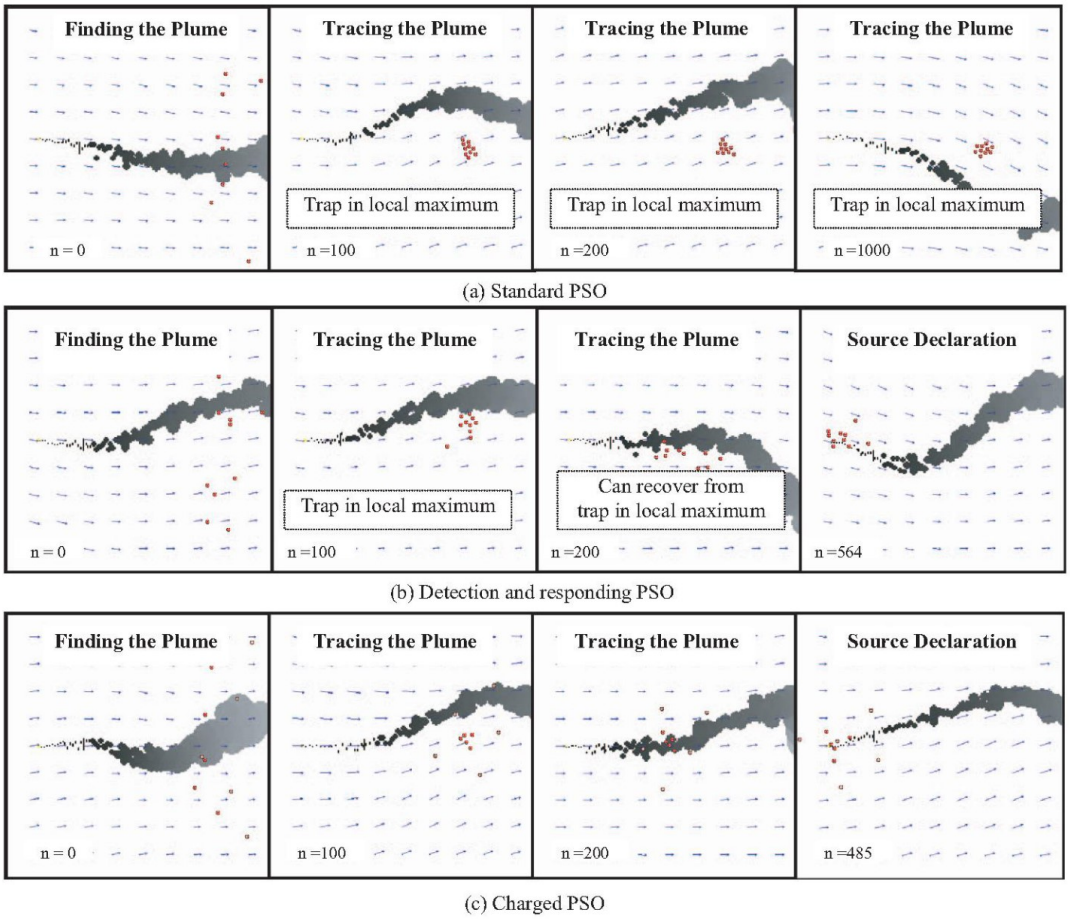


Figure 4.4: Comparison of S-PSO, DR-PSO and C-PSO with a dynamic (turbulent) OSL task [8].

PSO with Wind Utilisation

PSO with Wind Utilization (PSO-WU) incorporates wind measurements in order to improve performance [45]. Knowing the wind vector, the particles are prohibited to move to an area downwind as depicted by the 'forbidden area' in Figure 4.5. The size of the forbidden area is set by the angle $\theta_{forbidden}$. Another implementation of wind utilization multiplies the updated velocity vector with a factor between $[0, 1]$ based on the angle between the velocity and wind vector. This way, the particles are encouraged to move upwind where the odor source is likely located.

4.2.3. Glowworm Swarm Optimization

Glowworm Swarm Optimization (GSO) is a meta-heuristic optimization algorithm designed to find multiple optima in a search space [46]. Analogous to the glowworms in nature, agents emit light with a certain intensity to attract mates or food. In the case of GSO, the intensity of light is proportional to the local concentration value. Agents are programmed to move to a neighboring agent within their local decision range that has a higher luminescence value. After each movement step, the luminescence values are updated and the cycle is repeated. The advantages of GSO are that it can deal with highly non-linear problems and, compared to PSO, does not use velocity eliminating any overshoot/other velocity-related issues [47]. Another advantage over PSO is that all the agents only have to communicate locally with

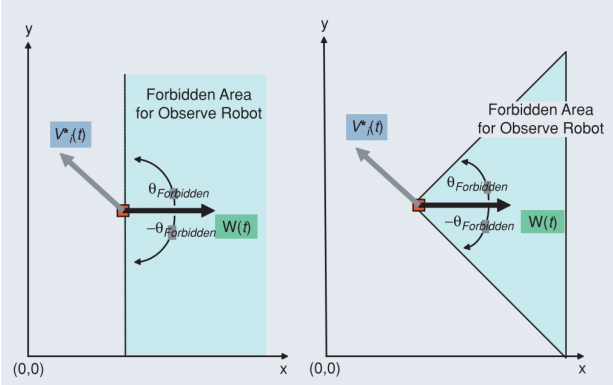


Figure 4.5: Implementation of PSO-WU by use of a forbidden downwind area [45].

only their neighbors and not the whole swarm, saving communication overhead. However, GSO is very susceptible to dynamic changes in the environment and would therefore be impractical to use in turbulent conditions.

Zhang et al. [48] created a Modified GSO (M-GSO) algorithm for GSL purposes. The method features four steps: plume finding, plume traversal, odor source declaration and forbidden area setting. When a gas source has yet to be detected, the agents conduct global self-exploration by randomly traversing the search area in spirals and zig-zags. If gas is sensed, the agents switch to a local GSO search tactic until they declare the source. With the source declared the region surrounding the source is designated a forbidden area to have the agents search elsewhere for other sources.

4.3. Probabilistic GSL Methods

Besides the development of the heuristic multi-agent approaches, research efforts are put into probabilistic methods. Most of these methods are based on Bayesian inference and variations on this framework exist such as the use of hidden Markov methods or the combination with PSO [13, 10]. As probabilistic methods are model-based instead of a search heuristic they can perform very well when using a proper model of the problem. Unfortunately, a good model still requires a few key assumptions that limit real-world applicability. For example, because these methods have to assume the gas plume to distribute in a certain way, they would fail in an environment with unknown obstacles as that would invalidate the earlier assumption about the gas plume distribution [12]. These methods are also susceptible to errors in wind measurement because the predictions can heavily depend on wind data depending on the model. Additionally, data about the chemical release rate has to be known/assumed although Bourne et al. devised a method to locate and estimate the plume source terms [11]. Moreover, these approaches require more computational resources compared to bio-inspired and heuristic methods, making them hard to deploy on smaller robots like micro UAVs.

Despite these limitations, probabilistic methods stay relevant through continuous improvements. This section first elaborates on the basic principle of Bayesian inference and how it is applied in GSL tasks. In the subsequent sections, notable variations such as infotaxis and entrotaxis are discussed.

4.3.1. Bayesian Inference

The process of Bayesian inference makes use of the fact that time-averaged gas concentration follows that of a Gaussian distribution in the direction of the flow [3]. With this knowledge, two probability maps can be constructed. First, given the location of the source, a probability distribution of the released odor molecules is derived. This is known as the forward algorithm. Also, a probability distribution of the odor source location is determined. Then, with the use of Bayesian inference, the inverse algorithm is created. The inverse algorithm iteratively updates the source location probability map based on (non-) detection events of the robot. Usually, robots are given a predefined path to scan the region of interest [20, 49].

4.3.2. Infotaxis

To improve the efficiency of the existing probabilistic GSL methods infotaxis is introduced by Vergasolla et al. [50]. Infotaxis is an information-driven search strategy, seeking to maximise the expected rate of information. It does this by striking a balance between exploration (moving in a certain direction) and exploitation (staying still to acquire more info in the same spot), see Figure 4.6. Another notable difference compared to methods based solely on Bayesian inference is that infotaxis employs the number of odour encounters in a given time rather than the binary (non-) detection. Compared to reactive methods, infotaxis performs better regarding search time in conditions with low chemical concentrations. However, infotaxis takes longer to find a source in conditions with higher chemical concentrations [34].

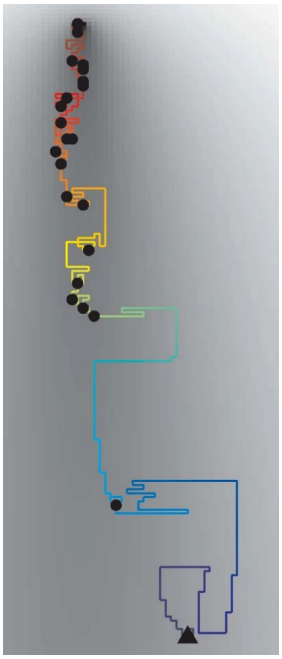


Figure 4.6: Typical infotactic trajectory in a windy (non-turbulent) environment. The triangle and points represent the start point and odor detections respectively [50].

4.3.3. Entrotaxis

Entrotaxis is a search strategy driven by maximizing the entropy of the predictive distribution [51]. Just like infotaxis, entrotaxis uses the number of odor encounters in a given time. However, entrotaxis calculates the possible number of odor encounters for every possible move based on the odor source distribution. From there the move maximising the entropy gain is chosen. Compared to infotaxis, entrotaxis has a similar success rate. However, the mean search time of entrotaxis is significantly lower compared to infotaxis, as are the computational resources required [51].

4.4. Machine Learning GSL Methods

GSL research has not been immune to the tremendous rise in popularity of Machine Learning (ML) recently. Although earlier applications of ML for GSL are limited, the advent of Deep Neural Networks (DNNs) and Deep Reinforcement Learning (DRL) result in some interesting findings [30]. ML implies that an algorithm learns to perform by means of training. The algorithm is usually represented by a neural network containing weights and biases. During training, these weights and biases are adjusted so that a given input (e.g. sensor data) results in the desired output (e.g. estimation of the gas source location). A big advantage of using neural networks and machine learning is that in theory any desired function can be constructed. Moreover, this function can be constructed automatically, eliminating the need for any prior knowledge of the problem. However, this does come with some disadvantages depending on the implementation which are discussed in the following subsections. For the purpose of this overview, this section is split into three common training methods; supervised learning, reinforcement learning and evolutionary algorithms.

4.4.1. Supervised Learning

Supervised learning refers to the use of labeled training data to fit an algorithm. A most rudimentary implementation involves a Feedworward Neural Network (FNN) fitted by backpropagation. During backpropagation, the output of the network is compared to the (correct) labeled training data resulting in an error. Subsequently, the partial derivatives of all the weights and biases over the error are calculated. These partial derivatives are then used to change the values of the weights and biases to improve the output of the FNN.

An advantage of such an algorithm is that it is relatively easy to implement. However, a sufficient quantity and quality of training data are required in order to train the algorithm properly. This results in research that deploys a static sensor array in the area of interest instead of a moving robot. This way, training data can easily be recorded by experimentation or simulation [52, 53, 54], see Figure 4.7. Additionally, this research often makes use of Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) to take advantage of the temporal data in gas measurements.

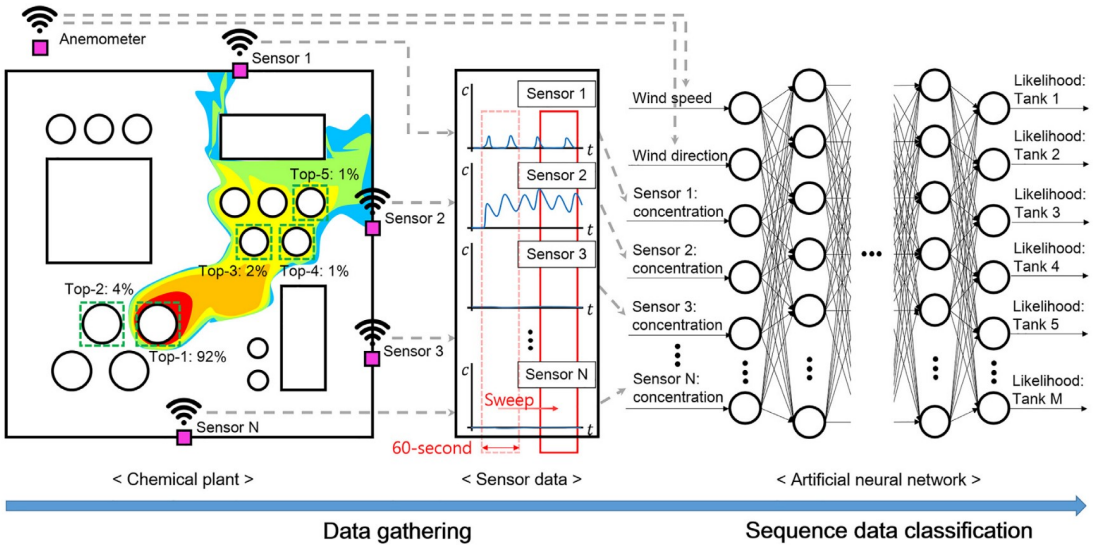


Figure 4.7: Schematic diagram of a localization model with a static array for a large chemical plant [53].

4.4.2. Reinforcement Learning

With Reinforcement Learning (RL), the algorithm is considered to be an agent that can interact with its environment through different actions it can take. The goal of the agent is to maximize a reward resulting from its actions.

An advantage over supervised learning is that RL negates the need for training data and can even learn in an online setting. This implies that an agent does not need to know everything about the environmental conditions or the odor source beforehand as it can learn while searching. Therefore, it has seen more use in research featuring agents that move and search for the odor source instead of a static array of sensors [14, 55, 56, 57]. A disadvantage of RL is that convergence is not guaranteed and that the result is sensitive to the reward and input parameters. Still, there is limited control over the learned behaviour of the agent because it requires non-trivial changes of the reward/value function.

4.4.3. Evolutionary Algorithms

An evolutionary algorithm (EA) is a metaheuristic optimization method that leverages natural evolutionary principles to improve their performance. It is characterized by the closed feedback loop between actuators (actions) and sensors (perception) through changes in its environment. This way, EAs learn about their complete system at once which is referred to as 'embodied cognition'. A basic setup of an EA is shown in Figure 4.8.

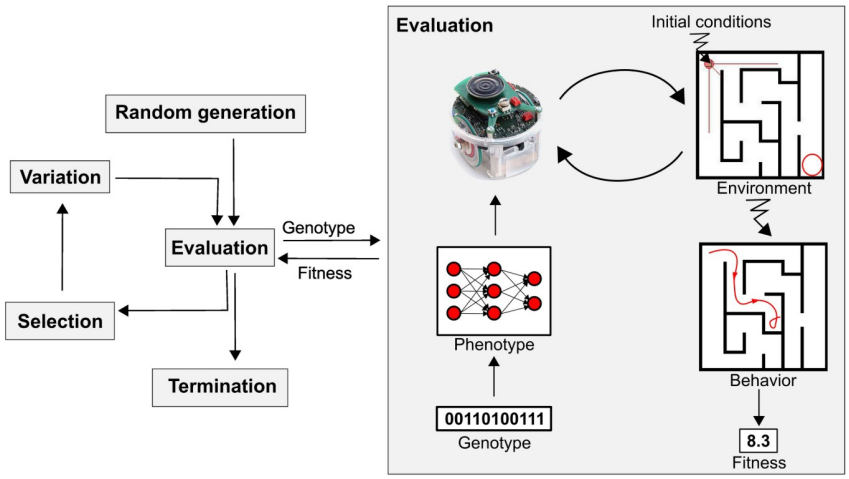


Figure 4.8: High level flowdiagram of an evolutionary algorithm [58].

Advantages of EAs are as follows. Compared to RL, EAs do not require a gradient to find a solution and are therefore less likely to get stuck. Moreover, due to the principle of embodied cognition, there is no limit on the amount and kind of parameters that can be optimized (sensors and actuators for example). Additionally, as any type of controller can be optimized with an EA there is control over the number of computational resources spent. Finally, no additional knowledge about the environment and how interactions would influence the environment is required.

A disadvantage of EAs, compared to RL algorithms, is that EAs are generally not capable of online learning and generally require more computational resources. Moreover, the chosen controller can limit the solution such that it performs unsatisfactorily. Furthermore, when a solution has not been found at all the cause is often unclear.

Evolutionary algorithms seem to be rarely used in GSL research. Nevertheless, there are some interesting implementations. Duisterhof et al. [5] employed an EA to evolve a PSO algorithm to find an odour source in simulation and experimented on real hardware with promising results. In earlier research, De Croon et al. [15] used an EA to train a Continuous Time Recurrent Neural Network (CTRNN) for GSL in simulation and analyzed its trained behavior, see Figure 4.9.

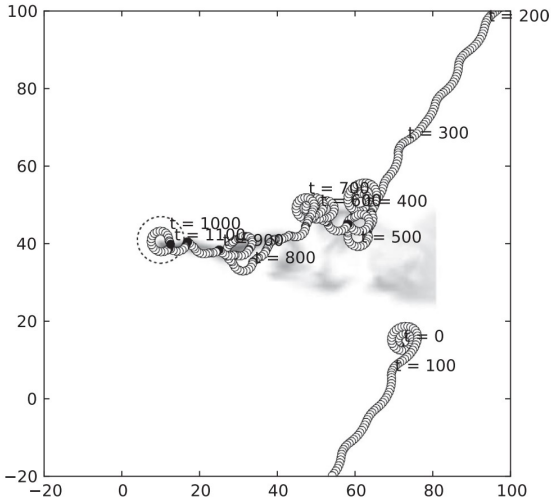


Figure 4.9: Best evolved agent localizes the gas source in turbulent conditions by De Croon et al. [15].

4.5. Method Related Simulator Requirements

The initial survey of GSL methods highlights some simulator requirements. Most importantly for all method types, the simulator must be able to offer the agent concentration data for a specific location and time (depending on the gas simulation type). For the performance measurement of methods that predict a complete concentration field as some probabilistic methods do, it is required that the simulator offers the complete ground truth concentration data. Furthermore, some methods utilize wind data to locate the source such as some variants of PSO, probabilistic and even ML methods. Therefore wind vectors should also be able to be passed to the agent.

Computational efficiency is always desirable. However, for multi-agent and ML methods it is vital to the practicality of the simulator. Multi-agent simulations require more and more resources with an increasing number of robots, especially if they use computationally expensive vision sensors. It is therefore important that the simulator is able to handle these agents efficiently, perhaps in parallel, to keep the simulation running at a satisfactory time factor. For ML methods, parallelization is of even greater importance. The generation of training data or the online learning process of an EA benefits greatly from increased computational efficiency and parallelization. This is because individuals of an EA can be evaluated separately. This is not necessarily the case for RL algorithms.

4.6. Method Comparisons in Literature

Analysis of GSL research regarding comparisons reveals a need for cross-category comparisons. Here, the categories correlate to this chapter's 4 sections above. From the 57 research papers studied, only 6 included comparisons with methods of a different category (e.g. a bio-inspired method versus a probabilistic method). Additionally, 12 papers featured no comparison to any other method, see Figure 4.10.

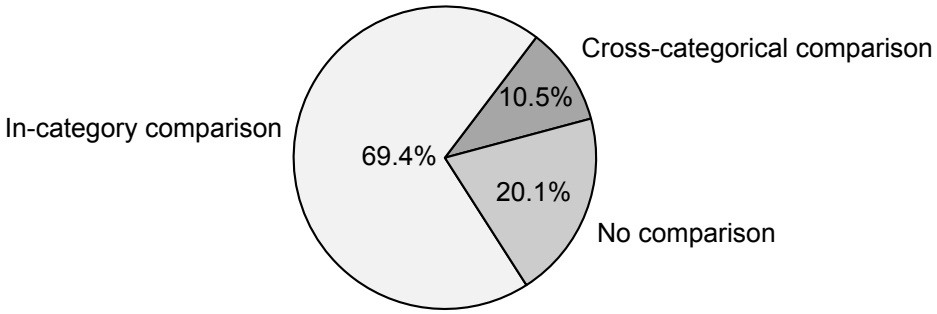


Figure 4.10: Percentages of studied papers in GSL research featuring in-category/cross-categorical comparisons. The categories are defined as follows: bio-inspired, multi-agent, probabilistic and machine learning.

Papers featuring cross-categorical comparisons share valuable insights. For instance, Duisterhof et al. compared PSO algorithms with varying parameters with chemotaxis and anemotaxis algorithms [5]. The bio-inspired methods performed on par with the PSO algorithm featuring manual parameters in terms of success rate, but they were outperformed by a PSO algorithm with evolved parameters in success rate, time and distance to the source. Voges et al. [34] compared three bio-inspired methods to infotaxis in terms of success rate, distance overhead, deviation and trajectory. Results show that infotaxis performs better in environments with low odor concentrations than bio-inspired algorithms. Prabowo et al. [12] combined a probabilistic strategy based on Bayesian inference with an anemotaxis strategy. Compared to three other strategies; entrotaxis, a purely Bayesian algorithm and a purely bio-inspired algorithm the combined strategy excelled in terms of path length. Unfortunately, no other commonly used performance metric (success rate for example) is used to compare performance. Finally, de Croon et al. [15] compared a CTRNN trained by an evolutionary algorithm to a silkworm moth strategy. They concluded that the CTRNN outperforms the silkworm moth strategy. As these papers have shown by their respective influence, cross-categorical comparisons are essential for progress in the research field of GSL.

Nevertheless, the majority of research presents in-category comparisons. Logically this is the most practical way to put the performance of a novel method into perspective. The depth and quality of comparisons vary across research. Some notable examples of quality comparisons are as follows. Russel et al. [7] compared two bio-inspired algorithms with a gradient-based method with simulation and experimentation. Performance was measured in terms of success rate, mean path length and standard deviation. Although not measuring performance, the most in-depth comparison of reactive strategies is done by Macedo et al. [59]. Their paper features an analysis of nine different methods (of which many are bio-inspired) from a state-action perspective. For each state, histograms of actions are built resulting in state-action mappings revealing common trends regarding strategy.

It is observed that each category contains a 'root' algorithm that is often used for comparison. For bio-inspired methods, those would be the E.coli or silkworm moth-inspired algorithms. Likewise, multi-agent methods are often compared with a standard PSO. Moreover, probabilistic methods usually feature a rudimentary method based on Bayesian inference for comparison. However, within the category of ML methods, there is no such common method.

Despite all the aforementioned comparisons, there is potential for improvement. Because of the differences in methods (assumptions, simulation/experimental setup etc.), the results of these papers are limited by the lack of objective comparison to other papers. This emphasizes the need for a common benchmark that can alleviate these limitations.

5

Robot Simulation Environments

The current state of simulation environments features a diverse selection. To make a simulator/benchmark accessible it is important to consider existing solutions, their popularity, advantages and disadvantages regarding GSL research. Some simulators are optimize for swift execution times while others require more computational resources to emphasize visual realism for example. This chapter first covers existing general-purpose simulation environments. Subsequently, different methods of gas and gas sensor simulation are examined. Finally, some GSL-specific simulators are presented.

5.1. General-Purpose Simulators

The success of a benchmark depends on the amount of adoption. To encourage this the software has to be accessible enough to persuade researchers into implementing their own method. In this case, accessibility is considered to be defined by three factors. Firstly, the availability of an API is taken into consideration. A simulator with APIs for common programming languages such as C++ and Python are considered. More importantly, the quality of documentation is taken into account. Lack of proper documentation hinders development. Lastly, the surrounding development community is important. For example, the presence of a forum where common questions get discussed and resolved or activity on GitHub issues are considered. Besides accessibility, practicality is reviewed by assessing the degree of realism, required system resources/execution time and potential API's for that simulator.

The remainder of this section discusses a selection of relevant existing simulators. Note that many other simulators have been considered such as Player/Stage [60], Webots [61], CoppeliaSim (formerly known as V-REP) [62], CARLA [63] and Unreal Engine [64]. Unreal Engine is interesting in particular due to its use in project AirSim [65]. Unfortunately, AirSim has since been deprecated.

5.1.1. Gazebo

Gazebo is an open-source simulator that started development in 2002, making it one of the more established pieces of software [66, 67]. Its development was initiated by the need for a high-fidelity outdoor dynamics simulator for Player/Stage, an older robotics simulator [60]. Currently, it is curated and maintained by the Open Robotics Foundation in collaboration with a community of developers. Note that there are two versions of Gazebo, Gazebo Classic ($\leq$ ver. 11) and Gazebo (which was shortly named 'Ignition' in 2019-2022). Gazebo Classic will reach its end of life in 2025, therefore it is important to use the latest Gazebo for continued support. Accessibility is good, as Gazebo's main API language is C++, a commonly used language in robotics. Moreover, contributors are actively assisting users with issues on GitHub. This is partly due to the fact that Gazebo is frequently used in combination with ROS, the open-source Robot Operating System [68].

Gazebo's main physics engine is DART [69] which features multiple collision engines like Bullet and ODE. Although DART is generally stable and accurate, it is relatively slow compared to other engines for the same accuracy [70]. However, if a simulated robot uses cameras for navigation it must be noted that photorealism is lacking in Gazebo. Additionally, Gazebo is less well suited for RL research as it can become unstable when sped up [71]. Required system resources highly depend on the size and complexity of the scene, number of agents and localisation algorithm. Nevertheless, Gazebo can be considered to be average regarding necessary system resources. For RL environments in Gazebo with ROS there is

DeepSim [72].

5.1.2. Unity

Although Unity is primarily a game engine it finds its use in GSL research as well. Therefore, it has a focus on photorealism and interactions with the environment. For this reason, Ojeda et al. created an environment in Unity for unmanned aerial vehicles (UAVs) to visually search for an odor source [26]. Unfortunately, Unity is not open-source and only supports C# as an API language which is less common in the field of robotics. Nevertheless, the documentation of Unity is excellent and the surrounding community is substantial.

Although the visuals of Unity are great, it requires a considerable amount of computational resources compared to the other simulators discussed. Physics simulation is therefore often handled separately. Interesting examples of this are Flightgoggles [24] and Flightmare [73]. Finally, Unity features an ML toolkit to enable the training of agents in Unity environments [74].

5.1.3. Swarmulator

Swarmulator is a lightweight, open-source, simulator ideal for prototyping spatial (2D) swarm behaviour [75]. It was created in-house by researchers at the TU Delft MAVlab. Written in C++, execution is swift and each agent is handled with their own thread, making them able to handle asynchronously [28]. Compared to Gazebo and Unity, Swarmulator is a niche piece of software. Therefore, although still sufficient, there is less documentation and there is no active development on GitHub.

Swarmulator excels in the simultaneous simulation of multiple agents. Visually, however, Swarmulator is flexible to generate basic imagery as required but lacks integration with a proper rendering engine. Because the code is relatively barebones and low level, one is free to implement any behavior or machine learning algorithm as they desire.

5.1.4. Isaac Sim™

Isaac Sim [76] is a simulator by NVIDIA that started development in 2019 and was officially released in 2021. The simulators' main focus is on minimizing the sim2real gap. Therefore it is capable of real-time photorealistic rendering, simulating high-fidelity physics and sensors. Originally, Unreal Engine and Unity were used for rendering before their integration depreciated. Now, NVIDIA's Omniverse™ is used for environment rendering. Additionally, the physics simulation can be run on the Graphics Processing Unit (GPU) resulting in a significant performance increase while being accurate. In addition to this, Isaac Sim features integration with different ML tools.

As Isaac Sim is a relatively new development, adoption is still limited. Nevertheless, there is good documentation with tutorials in addition to a dedicated forum. APIs for ROS/ROS 2 and Python are present and there is even a Software Development Kit (SDK) to create additional applications and plugins for Isaac Sim. This set of features and community makes the simulator suitable for continued development in the future. Unfortunately, Isaac Sim is closed-source and relatively new, therefore adoption might pose a challenge. Moreover, the development of Isaac Sim is entirely dependent on the NVIDIA Corporation and its Omniverse environment.

5.2. Gas Dispersion Simulation

Simulating the complex phenomenon of gas dispersion poses a great challenge in GSL research. The process must match real-world behavior as much as possible while still being computationally efficient enough to be practical. Nevertheless, proper simulation brings advantages that extend to experimentation as a whole. Ground truth information is easy to obtain with simulation. Moreover, simulation offers perfect repeatability. Especially with gas dispersion, this is important because small variations in parameters have a major impact on resulting datasets. Lastly, real-world data acquisition is time-consuming. A simulation has therefore the potential to increase the efficiency of research. This section covers commonly used gas dispersion models, from real-world measurements to CFD analysis.

5.2.1. Real-World Dispersion Measurements

Before the advent of proper simulation real-world measurements were the only way to gather information on the process of gas dispersal. To the present day, real-world measurements stay relevant for the validation

of novel dispersion models. For example, Farrel et al. [77] and Monroy et al. [25] used experimental data to validate their filament-based plume shown in Figure 5.3. Other research by Cabrita et al. [27] validated the simulation of their meandering gas model with real-world measurements. Unfortunately, validation of simulation by real-world experimentation is uncommon.

Real-world data can also be used in simulation directly. Jones [4] conducted real-world experiments to formulate numerous statistical parameters to model the experimental observations. Experiments were conducted in a wind tunnel using negatively ionized air as a tracer. He concluded that the plume structures feature high levels of intermittency and peak-to-mean ratios. Additionally, the puffs in a plume are generally of small cross-sections. Therefore, these puffs feature a high concentration and the mean of concentration is acquired by consecutively swift bursts of intense concentration. Finally, fast sensor response time is vital to determine the proper statistical parameters of the plume structure. This was tested by applying a low-pass filter to the sensor. Likewise, Webster et al. [78] determined statistical plume characteristics by exploiting Laser-Induced Fluorescence (LIF) in liquid, see Figure 5.1.

Figure 5.1: Laser-induced fluorescence measurements of a turbulent plume by Webster et al. [78]. The plume is released isokinetically close to the wall into a fully developed boundary layer.

5.2.2. Mathematical Gas Dispersion Models

Mathematical models can be divided into box models, Gaussian models, Eulerian and Lagrangian models, filament-based models and Computational Fluid Dynamics (CFD) [79, 80]. Because CFD will be elaborated upon more than the aforementioned models, it is treated in Section 5.2.3.

Box Model

The box model is a zero-dimensional model that follows the principle of the conservation of mass. The box is considered a virtual region in which particles can undergo chemical and physical processes. Moreover, particles are free to move in and out of the box. Inside the box, the air is considered to be well-mixed and uniform. The model is useful for simulating detailed chemical reactions and aerosol dynamics. However, the box cannot provide any local concentration information, making the box model inadequate for the simulation of highly dynamic particles influenced by wind and turbulence.

Gaussian Dispersion Models

Gaussian models are a common method to simulate gas dispersion and assume that gas dispersion follows a normal probability distribution, see Equation 5.1 and Figure 5.2 (a).

$$\bar{C}(x,y,z) = \frac{Q}{2\pi\sigma_y\sigma_z\bar{u}} \exp\left(-\left(\frac{y^2}{2\sigma_y^2} + \frac{z^2}{2\sigma_z^2}\right)\right) \tag{5.1}$$

Here, $\bar{C}(x,y,z)$ depicts the average concentration as a function of location in x, y, z. Q is the chemical release rate and $\bar{u}$ is the mean wind speed. The amount of dispersion of the plume in y and z is determined by σ_y and σ_z respectively. These coefficients are defined by the distance from the source in addition to atmospheric stability and wind speed.

The analytical nature of the Gaussian model requires fewer computational resources than its numerical counterparts. However, its practicality is limited by the fact that is a steady-state approximation. This implies that the wind field is assumed to be uniform and the plume behavior will not change over time. Therefore, no recirculation or turbulent effects can be modeled that would be present in complex environments featuring buildings, walls or other (perhaps moving) obstacles. Variations on the Gaussian model have been introduced to resolve some of these limitations for specific use like the meandering plume model shown in Figure 5.2 b.

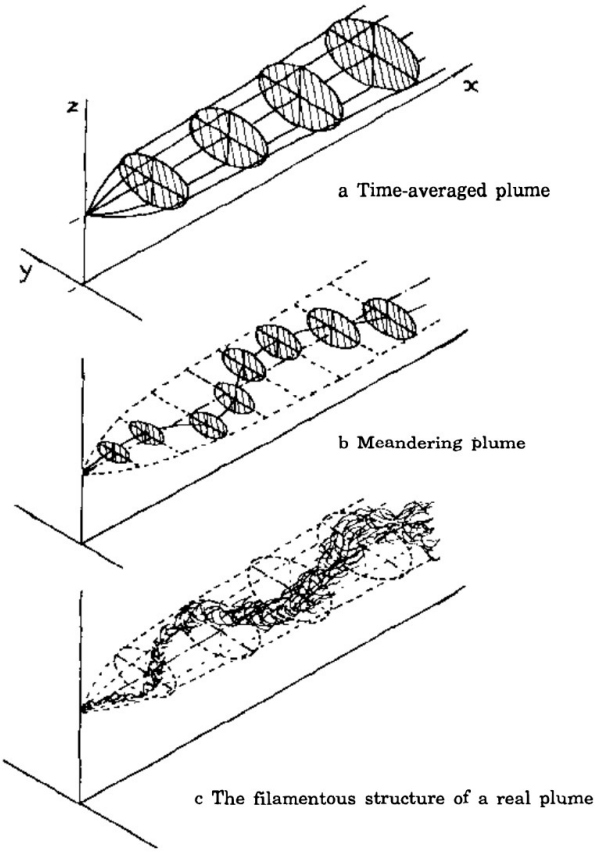


Figure 5.2: Three different kinds of plumes by Murlis et al. [81]. A Gaussian, time-averaged plume (a). A meandering plume model with the distributions centered around the sinusoidal meandering line (b). The structure of a real plume (c).

Eulerian & Langrangian Dispersion Models
Eulerian and Lagrangian models describe dispersion with dynamic advection and diffusion equations. The models differ in their frame of reference. While the Eulerian model employs a fixed frame of reference (usually a grid system with orthogonal coordinates), the Lagrangian model utilizes a reference frame following the average wind trajectory. This implies that for a Lagrangian model, the governing equations can neglect the terms for advection i.e. the movement of particles along the wind direction.

Like Gaussian models, Lagrangian models cannot handle environments resulting in complex wind fields. Moreover, the implementation of sources and sinks is hampered by the assumption that air parcels must stay intact. Nevertheless, Lagrangian models are less computationally expensive than Eulerian models [80].

The practicality of Eulerian models is limited by the assumption of homogeneity within each grid cell. Therefore, no extra information can be resolved at the sub-grid cell level. To alleviate this limitation the grid cell size can be reduced to simulate in more detail. This approach can become computationally expensive quickly.

Filament-based Model
Farrell et al. [77] introduced a filament-based model to efficiently simulate gas dispersion in turbulent environments. Unlike the previously mentioned methods, the filament-based model is a consolidation of different dispersion phenomena resulting in an accurate replication of short- and long-term exposure statistics. The model simulates the release of gas as puffs containing filaments of normally distributed odor concentration, see Figure 5.3. Three dispersion phenomena allow the plume to realistically evolve over time. Eddy structures (the swirling of the wind field) larger than the puff size move the puffs as a whole, simulating advection. Eddies approximately matching the puff size cause the puff to considerably

distort and alter the motion of the filaments. Lastly, smaller eddies mix the filaments in the puff with little distortion of the puff itself. Do note that the filament-based model required a wind field as input. For simple environments, this wind field might be generated by a simple mathematical function. In more complex environments, however, the wind field data is usually generated using CFD.

The filament-based model strikes a balance between realism and required computational resources. The implementation allows for dispersion simulation in complex (indoor) environments featuring obstacles. However, the model does not take into account the effects of gravity or temperature gradients on the plume, among other complex phenomena that may influence the dispersion.

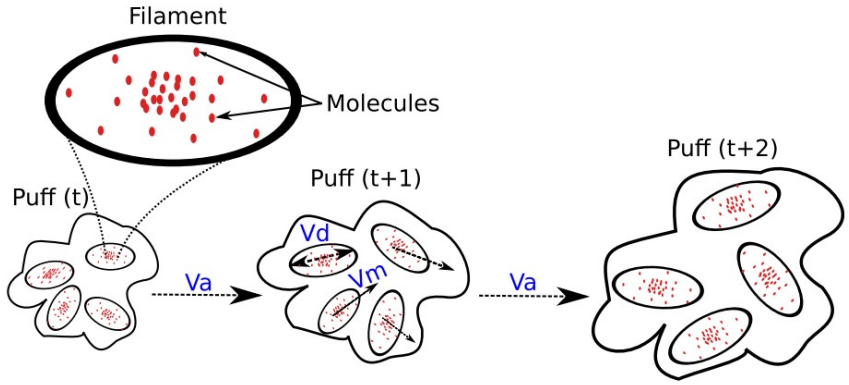


Figure 5.3: Filament-based plume model featuring puffs containing filamentes. Within each filament, the odor concentration (molecules) is normally distributed. Large eddies cause puff advection (V_a), while medium and small eddies in the wind field cause the filaments to mix and distort (V_m and V_d) [25].

5.2.3. Computational Fluid Dynamics

Computational Fluid Dynamics (CFD) is a numerical method that allows solving for many parameters such as wind velocity, temperature and gas concentration depending on the model. Due to the increase in available computational resources it has become a popular method for simulating gas dispersion. The general workflow of a CFD simulation is as follows. First, a model of the region or object of interest is used to generate a mesh. Mesh generation divides the environment to be simulated (the air) into smaller, discrete parts. A proper mesh is important as it serves as the approximation of the larger domain. In regions where the properties of the flow (pressure, density etc.) feature a high gradient the mesh size is smaller to ensure an accurate solution.

Subsequently, boundary conditions are introduced. For example, an open window letting air in or out is considered a source or a sink respectively. Additionally, the boundary condition of a wall implies that there is no slip at the surface of that wall. After selecting a solver, miscellaneous physics settings and convergence criteria the algorithm tries to solve the model until the convergence criteria are satisfied. This can take multiple tries, as sometimes the solver does not manage to converge properly. Finally, the resulting data can be used in robotics simulation for GSL tasks. CFD puts out realistic results but can not be run online like similar methods. Usually, CFD is performed beforehand.

5.3. Gas Sensor Simulation

Apart from a simulation engine and the type of gas plume simulation, variate simulation methods for gas sensors are used in research. From the 40 papers studied, 17 tested their GSL method without the simulation of a sensor. This implies that the instantaneous gas concentration present is directly passed to the localization algorithm. This is a straightforward approach which might be valuable in an isolated case of comparing different methods and assessing their performance. Nevertheless, the lack of sensor dynamics increases the sim2real gap, thereby hindering the transition to real-world experimentation. Therefore, different methods of sensor simulation are proposed. In this section, they are divided into custom implementations and simulation of real-world sensors.

5.3.1. Custom Sensor Implementation

Custom implementations usually rely on a combination of either a concentration threshold value and/or noise. Jatmiko et al. [8] tested a PSO algorithm with a sensor model given by Equation 5.2.

$$S(t) = \begin{cases} C\left(\left[\frac{t}{\Delta t}\right]\right) + e(t) & \text{If } C > \tau \\ 0 & \text{Otherwise} \end{cases} \tag{5.2}$$

Where $S(t)$ is the sensor response, C is the gas concentration and e is random sensor noise with $e << C$. Moreover, the sensor only reads when the concentration is above a set threshold τ. Finally, $\left(\left[\frac{t}{\Delta t}\right]\right)$ is used to resolve the discrepancy in time step resolution between the robot and the gas simulation. Although this implementation is one step beyond the direct reading of gas concentration in logical development, it only improves by adding noise and a threshold value.

Song et al. [16] conducted their research on collaborative infotaxis using a binary sensor implementation. For a specific time interval and location the sensor either reads no concentration or any concentration which are represented by a zero or one respectively. Over time, the sensor reads odour counts rather than concentration. This combination of detection events and trajectory is often used for probabilistic GSL methods. This is because it simplifies the input for the subsequent inference algorithm. However, this approach poses problems when transferring to the real world. For example, Moraud et al. [82] conducted real-world experiments with a ground-based robot and an infotaxis GSL method. After extensively testing several odour sensors they concluded that their sensitivity and response time is inadequate for an infotaxis scheme. To solve this, they resorted to the use of a temperature sensor that quickly reacts and does not saturate easily to detect a source of hot air.

5.3.2. Real-World Sensor Simulation

Simulating real-world sensors is the most elaborate but realistic form of sensor simulation. Li et al. [10] and Meng et al. [83] simulate a Metal Oxide Semiconductor (MOS) sensor to use in their simulation. The physical working principle of a MOS sensor expresses itself as a lagged response and recovery. To simulate this two separate second-order lags are implemented as shown by Figure 5.4 and Eqs. (5.3) and (5.4). Moreover, Meng et al. [83] include Gaussian noise to further increase realism. Additional parameters that can be added to this model are sensor bias and possible drift. Although the implementation requires extra computational resources it is invaluable to bridging the sim2real gap.

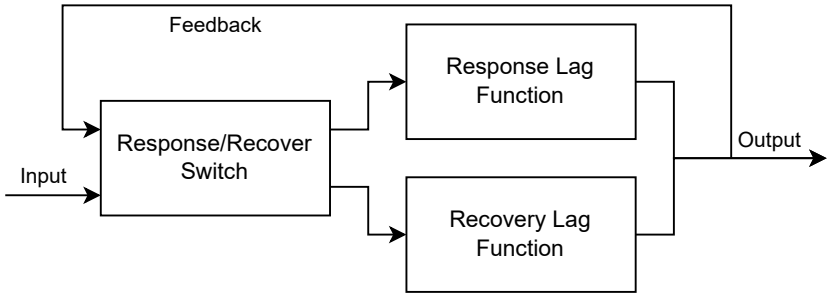


Figure 5.4: Block diagram showing the dynamics of a simulated MOS sensor response. The phase switch switches to the response/recovery blocks based on an increase or decrease in the input concentration. The response and recovery dynamics are a second-order lag given by Equation 5.3 and Equation 5.4 [10, 83].

$$\frac{y_{res}(s)}{x(s)} = \frac{1}{(1 + A_{res}s)(1 + B_{res}s)} \tag{5.3}$$

$$\frac{y_{rec}(s)}{x(s)} = \frac{1}{(1 + A_{rec}s)(1 + B_{rec}s)} \tag{5.4}$$

Likewise, Monroy et al. [25] include the simulation of a MOS and a Photo Ionisation Detector (PID) sensor in their developed gas dispersion simulator GADEN. To accomplish this they consult the technical data sheets provided by the manufacturers. For the MOS sensor, this implicates the calibration of the right sensor resistance given a reference resistance and the use of a low pass filter as discussed earlier. Compared to a MOS sensor, a PID sensor shows a quicker response. However, it is unable to distinguish between different gasses. Rather, it provides the total concentration of all gasses present. The PID sensor behavior is also simulated according to the provided technical specifications. Finally, validation experiments were conducted to test the accuracy of the plume and sensor simulation.

5.4. GSL Specific Simulators

Apart from the general-purpose simulators discussed earlier, the majority amount of GSL research is performed with custom simulations developed in-house. Awadalla et al. present an example of a combination that is often used; MATLAB and Ansys Fluent (both paid software) [18]. In their research, they developed a framework capable of simulation in three dimensions. First, they model the environment in Ansys and simulate a release of gas using CFD in an indoor setting. They assume certain windows and doorways to be the inlets and outlets to create a slight breeze. This situation might not be realistic but tailors well to the presented case study. Finally, the resulting information is passed to the MATLAB simulation code.

Duisterhof et al. created an automated pipeline for environment generation called AutoGDM and combined it with Swarmulator for the simulation of multiple agents [5]. The generation of environments with AutoGDM consists of three stages as shown by Figure 5.5. First, a room is randomly filled with walls and corridors resulting in an occupancy grid map. This map is transformed into a 3D CAD (Computer Aided Design) model to pass it to the CFD simulation. CFD is performed with OpenFOAM. It first generates a mesh from the imported CAD model and then calculates a flow field for the given boundary conditions. This flowfield is handed to GADEN, a gas dispersion simulation, which simulates the gas plume with a filament-based model [25]. Swarmulator is then used to simulate multiple agents searching for the gas source location. This approach is specifically tailored to quickly generate multiple environments with different complexities. This way, an ML algorithm can progressively learn to locate the gas source in increasingly difficult environments.

Finally, Cabrita et al. implemented a GSL simulation called PlumeSim in the Player/Stage simulation environment [27]. The simulator is capable of using different inputs to generate a plume; from real-world data, a CFD simulation or a custom mathematical model. Two mathematical models, a Gaussian and a meandering model are already implemented. Additionally, they argue that in the field of OSL research, more emphasis has to be put on gas sensor modeling as it greatly affects the performance of the simulated algorithms and can increase the realism of the simulator. They introduce a sensor model of a MOX (Metal Oxide) sensor featuring rise time and decay time. The parameters for this model are collected through real-world experimentation. Unfortunately, Player/Stage has become obsolete therefore this GSL-specific simulation setup is not used anymore.

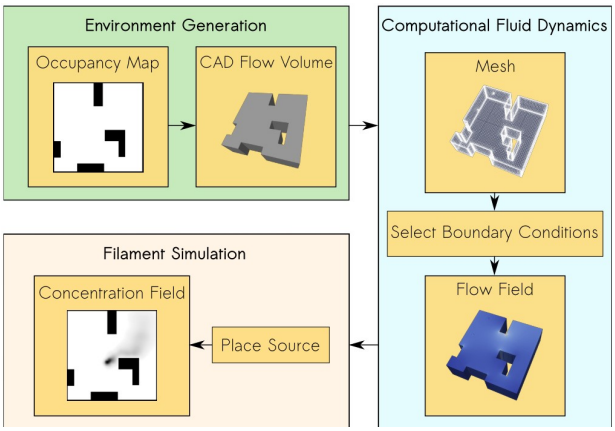


Figure 5.5: Flowchart of environment generation with AutoGDM by Duisterhof et al. [5].

5.5. Conclusion and Discussion on Simulation

A diverse selection of general and GSL-specific simulators is studied to consider existing solutions, their advantages and disadvantages regarding GSL research. The choice of a simulator will ultimately be decided after some initial experimentation. The most widely used option is Gazebo in combination with ROS and GADEN. As Gazebo and GADEN are well established in the research field they present a good option for creating a benchmark that is widely accessible. However, Gazebo can become computationally expensive and is not photorealistic. These shortcomings hinder large-scale online Machine Learning (ML) and the simulation of vision sensors. For a more computationally efficient simulation, Swarmulator is a good option. In combination with AutoGDM and GADEN, it presents a useful tool for GSL and ML. However, its lightweight implementation lacks any realistic visualization thereby excluding the use of vision sensors. Therefore Swarmulator is considered too impractical to use in further testing. Compared to Gazebo and Swarmulator, Unity provides excellent visual fidelity. Because it is originally developed as a game engine the physics is usually handled separately for accurate simulation. Flightgoggles [24] and Flightmare [73] are good examples of this. Moreover, the availability of excellent documentation and support in combination with the use of C# make this simulator one of the most accessible options considered. Lastly, a relatively new simulator to consider is NVIDIA's Isaac Sim. The focus of this simulator lies in minimizing the simulation to reality gap by the use of photo-realistic rendering and accurate physics. For accessibility, it features APIs for ROS and Python and features an ML toolbox. Unfortunately, Isaac Sim is closed-source and relatively new, therefore adoption might pose a challenge. Moreover, only future experimentation with Isaac Sim will elucidate the challenges of implementing GSL methods and plume simulation in the simulator.

From the analysis of gas simulation methods, it can be concluded that the filament-based model in combination with CFD simulation is the most relevant for future GSL simulation. Other simulation methods such as the box model or the Gaussian plume model make use of assumptions that are considered to limit realism and practicality. The combination of CFD and a filament-based model allows the use of accurate wind and dispersion fields in complex environments featuring multiple obstacles. One downside however is the fact that the CFD simulation cannot be run in an online fashion. Nevertheless, this is considered a worthwhile tradeoff for the realism it introduces.

Regarding sensor simulation, the following can be concluded. An important basis for all sensor simulations is the availability of accurate instantaneous gas concentration data. This will require high-fidelity gas simulation both in a spatial and temporal manner. From there, multiple sensor types can be implemented such as a MOX or PID sensor. Furthermore, on top of these simulated sensors, other representations such as a binary or averaged reading can be applied to suit the method at hand.

6

Real World Experimentation

A simulated benchmark is more valuable if it represents real-world experimentation which can be used to validate simulated methods. To achieve this, the current state of experimentation is surveyed. Special attention is brought to the arena size of the experiments, the presence of obstacles, the (artificial) wind conditions and the robot type. Moreover, some trends regarding these characteristics are observed. The chapter concludes with a discussion of future real-world experiments mimicking practical applications of GSL robots.

6.1. Current State of Experimentation

Approximately half of the studied GSL research papers conducted real-world experiments. The majority of these experiments are carried out in relatively small arenas [84, 21, 19]. The discrepancy between arena sizes is illustrated by Figure 6.1. The use of small arenas has multiple advantages besides the fact that it requires little space. If the experiment serves as validation of a simulation then the simulation does not have to be as large thereby limiting the required computational resources. Moreover, with a small arena, it can be argued that the experiment focuses more on the actual localization of a gas source rather than the exploration of an environment. Nevertheless, it is more logical to consider exploration an important phase of a GSL task. Note that some arenas have substantial lengths compared to their width as can be seen in the bottom of Figure 6.1. These experiments were mostly conducted in wind tunnels resulting in conditions that do not require any exploration at all [6, 40, 22, 2]. In these cases, plume tracing rather than plume acquisition and exploration are tested.

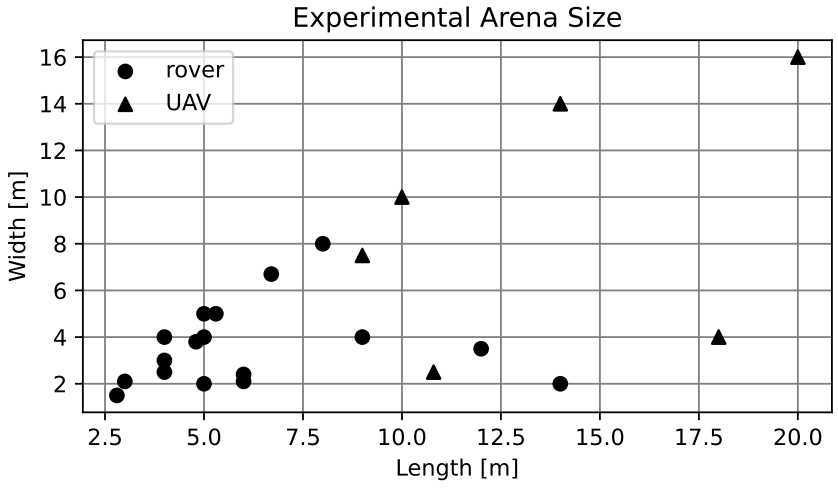


Figure 6.1: Dimensions of the experimental arena's studied throughout literature. The majority of experiments feature a rover and a smaller arena.

A ground-based robot is used most often compared to (micro) UAVs. The advantage of ground-based robots is that they are not restricted by weight. This expresses itself by being able to have longer endurance

and more elaborate sensor suites on board. Additionally, ground-based robots are able to use their wheel odometry data (often noisy) for localization in addition to other means such as ultrasound, LiDAR or GPS sensors.

Almost all of the experiments feature an artificially created wind field, while some introduce no wind field and test for diffusive conditions [49, 52]. Usually, a wind field is created using a desk or computer fan. Outdoor experiments are often carried out in low wind conditions to limit the turbulence present. Although these setups are convenient, they do not resemble a realistic scenario. Ultimately, apart from the GSL methods used, it can be stated that the current arenas can be considered to be rather straightforward challenges for the robot due to the combination of small arenas without obstacles and artificial (favorable) wind fields.

6.2. Experimentation Trends

GSL experiments are becoming increasingly complex. More recent research introduces larger (outdoor) arenas, see Figure 6.2. Additionally, an increasing amount of research features obstacles in their arenas, thereby further increasing the difficulty. The main reason environments are getting increasingly difficult is the fact that researchers acknowledge that experimentation in small, simple environments is of limited practical relevance. Additionally, GSL methods are improved to handle unknown environments and obstacles [12].

Furthermore, the popularity of (micro) UAVs for GSL tasks is increasing [5]. Recent developments in UAV technology make the platform more accessible and they prove to be versatile for exploration and GSL tasks. Nevertheless, the use of micro UAVs is challenging due to their weight restrictions. Therefore, onboard sensors have to be limited in weight and power consumption, further increasing the difficulty of a GSL task.

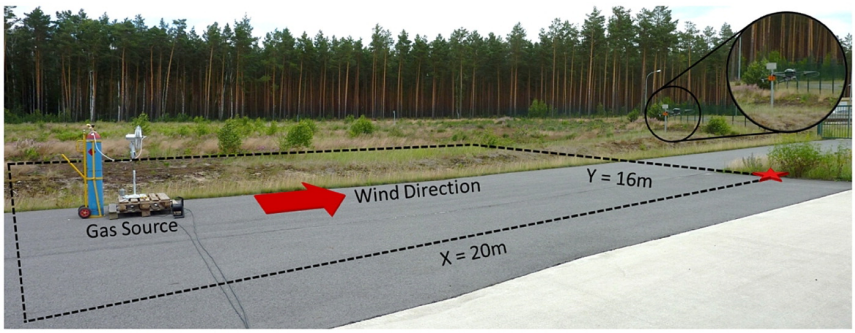


Figure 6.2: Outdoor experimental setup by Neumann et al. [85] featuring a micro UAV with a methane source on the left side.

6.3. Discussion on Future Experimentation

Experiments are expected to get more and more complex to the point that they can fully emulate practical real-world scenarios. It is therefore important for a benchmark to offer the flexibility to develop into something that can simulate it. In other words, the simulator must offer room for environments and situations to grow in complexity to the point of practical real-world experiments.

A few examples of 'practical' experiments are given. Firstly, consider a large indoor warehouse or factory. Having rovers or UAVs be able to swiftly cover a large area looking for a gas leak reduces the risk of casualties. Moreover, a UAV could be invaluable to search for odors that rise to the ceiling of the building. Another practical scenario is the surveillance of a large outdoor chemical plant. Or imagine the supervision of major roadworks. In these scenarios, there is always a risk of striking a gas pipe. Lastly, but maybe most importantly, GSL robots can assist in disaster relief. Especially with the use of UAVs substantial amounts of an area can swiftly be covered regardless of the terrain conditions.

The capability to easily change environment complexity in the simulator will be a key feature for training and benchmarking. Scenarios can feature different levels of complexity in terms of environment type and size, wind (gas) conditions and obstacles. For example, an indoor space might be a large rectangular warehouse or factory floor, or an office building featuring hallways and many rooms. The gas conditions

can vary widely as well. In most indoor scenarios the wind field is probably negligible resulting in diffusive conditions. Or maybe there is some kind of ventilation present. This can be an air conditioning system/open window or, in the case of a large hall, an open garage door. The buoyancy of the gas also influences the difficulty of the task. Finally, the type and amount of obstacles affect the task difficulty. A factory/warehouse floor might be open or filled with storage and machines. The same concept can be applied to obstacles in office buildings and outdoor environments.

7

Performance Metrics

Throughout GSL research a wide variety of performance measurements is used. The percentage of successful runs and (average) localization time are the most commonly used metrics. These elementary metrics combined can give a good indication of an algorithm's performance. Nevertheless, more involved types of measurements may offer other valuable insights into an algorithm's performance. This chapter presents commonly used metrics to discuss their practicality.

7.1. Successful Runs

Successfully completed runs is a widely used metric that mainly describes an algorithm's reliability. For this metric to be statistically meaningful it is important to perform a sufficient amount of runs. The amount of successful runs does not however indicate how 'well' the algorithm completed the GSL task in terms of other metrics such as search time/steps or movement overhead for example. Therefore, caution should be taken when interpreting the number of successful runs. For example, if various algorithms have high success rates it might indicate that the experiment lacks difficulty. This problem can be mitigated by measuring the success rate when no odor source is present.

An example of good use of success rate is shown by Voges et al. [34]. In their research, they perform multiple trials (ranging from 18 to 73 trials) for each different method and concentration. This amount of trials is considered to be sufficient to gain significant statistical information. Furthermore, the testing of different conditions and methods results in valuable comparisons, see Figure 7.1.

Less practical use of the success rate metric is presented by Ercolani et al. [2]. In their research, they use micro UAVs to locate an odor source at the end of a wind tunnel. They conducted five trials for two different sensor configurations: mounted on top or on the bottom of the UAV. As the sensor configurations resulted in a success rate of 100% and 80% respectively it would be interesting to see the success rate when no odor source is present.

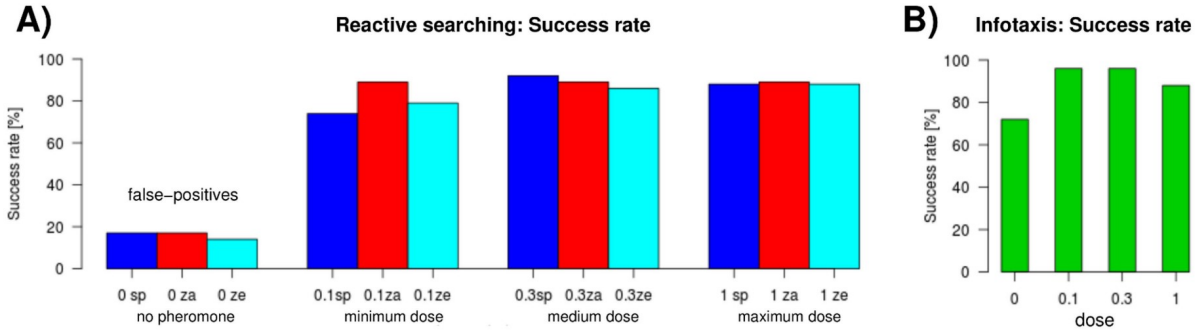


Figure 7.1: Success rates of reactive and infotaxis searching methods by Voges et al. [34]. Three reactive strategies are tested with different concentrations of the odor source. They are: Spiraling only (sp), arithmetic spiral & zigzagging (za), exponential spiral & zigzagging (ze).

7.2. Search Time & Steps

Search time is usually expressed in simulation runtime or real-time for experiments. Search time is closely related to the search steps which indicates the number of iterations the algorithm requires to find the source. Individually, these metrics are only useful when they can be compared with the search time and steps of other algorithms [86]. Therefore they are usually combined with other metrics to get a better picture of the performance.

Nevertheless, search steps can be an indication of the 'efficiency' of a method. For example, if it takes a novel PSO algorithm half as many iterations to find the odor source than an older PSO method it can be said that the novel algorithm is more efficient. Still, steps as an efficiency metric must not be confused with the required computational resources of a method. It could be that a method requires a few steps to locate an odor source, but each step may be computationally intensive.

7.3. Distance & Movement Overhead

The covered distance is not often presented as a metric but is still valuable for comparison purposes. In the case of multi-agent methods, traversed distance can be averaged over the agents [5]. Covered distance in itself is more intuitive when the minimum required distance is taken into account. Therefore, movement overhead is often introduced. Movement overhead is described by the percentage of excess movement the agent performed to find the source. It is defined as $\alpha = d/\Delta - 1$, where d is the distance traveled and Δ is the straight line distance between the starting position and the odor source location [40].

The implementation of a movement overhead metric becomes more complex with the presence of obstacles in the arena. The 'straight line' distance will then have to be determined by a shortest path algorithm. On the other hand, the straight line distance can still be calculated as before for simplicity. In that case, it must be made clear that a perfect movement overhead score might be impossible because the line from the start to the odor source might pass through an obstacle.

7.4. Trajectory

In some cases, research evaluates the performance of an algorithm by looking at the trajectory of an agent. Pequeño-Zurro et al. [87] evaluate the performance of an algorithm by the trajectory directness metric $\bar{D}$. It describes the average straightness of a trajectory compared to a reference vector pointing from the start position to the source location, see Equation 7.1.

$$\bar{D} = \frac{1}{L}\left(\sum_{i=1}^{N} r_i \cdot \cos\theta_i, -i \cdot \sum_{i=1}^{N} r_i \cdot \sin\theta_i\right) \tag{7.1}$$

The path of the agent is divided into N vectors. The length and deviation of each vector are described by r_i and θ_i respectively. The total traversed distance L is defined by $L = \sum_{i=1}^{N} r_i$. The resulting trajectory directness $\bar{D}$ is a vector with a length between $[0, 1]$ and a heading between $[-\pi, \pi]$. The thought behind this metric is that a more straight, continuous track represents more stable performance. Although extensive analysis is performed in Pequeño-Zurro et al. [87] work, this metric it is only applied to evaluating plume tracing performance as the agent's starting position is located inside the plume. Logically, trajectories in a plume discovery phase are more exploratory thereby limiting the usefulness of this metric in experiments representing a larger GSL task.

In other research, the trajectory is analyzed in a broader sense rather than hard metrics. De Croon et al. [15] evaluates the trajectory of an ML method to investigate the neural network dynamics of the agent. Moraud et al. [82] observe the trajectories of an infotaxis scheme for different wind conditions. By plotting a heading histogram (w.r.t. the wind vector) they conclude that the agent has a tendency to move upwind in a similar fashion to moths.

Ultimately, trajectory metrics tend to be useful when describing the characteristics of the search method. For direct comparison of different algorithms, trajectory directness may be useful in the phase of plume tracing.

7.5. Localization Accuracy & Error

Accuracy and error metrics are often used when predicting the source location or concentration distribution. Therefore probabilistic methods are regularly evaluated using these metrics. In these cases, it is assumed that the agents will detect at least some gas concentration in order to predict the source location. Ferri et al. [20] evaluated the performance of a Bayesian grid mapping method of three sources by plotting the localization error, see Figure 7.2.

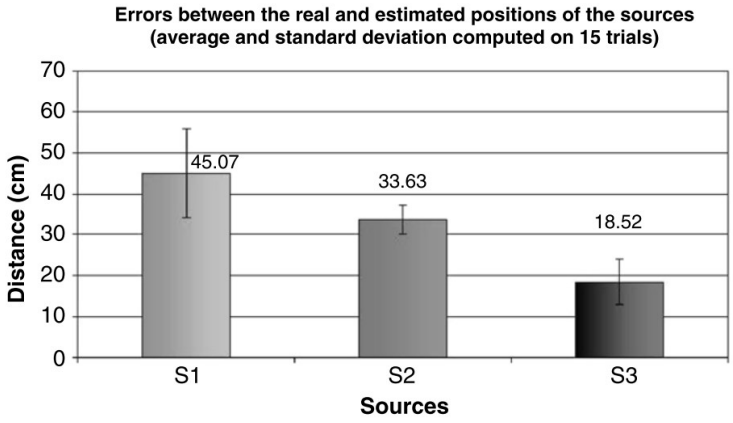


Figure 7.2: Estimation error of the source location for a Bayesian grid mapping method by Ferri et al. [20].

Additionally, a rarely used but interesting error metric is the Earth Mover's Distance (EMD) [49, 56]. The EMD calculates the separation between two discrete histograms or distributions. It can be viewed as the first Wasserstein metric's discrete counterpart. In other words: The EMD calculates the cost of converting one probability distribution into the other given two probability distributions that depict two distinct strategies of accumulating a certain probability mass (or earth). In other words, it calculates the least effort required to transfer mass from the first distribution to the second distribution, where the effort is equal to the mass's quantity times the mass's transfer distance. This approach is limited to the availability of the gas concentration ground truth of the entire search space, making it impractical for real-world experiments

7.6. Conclusion and Discussion on Performance Metrics

To conclude this chapter on performance metrics some final observations are made. Success rate, movement overhead and search time metrics are most often used and are proven measurements to assess GSL performance. The benchmark will therefore feature these metrics for all methods and environments. Furthermore, other metrics might be of interest depending on the method used. The performance of probabilistic methods might be assessed by accuracy and Earth Mover's Distance (EMD). Additionally, ML methods might be better characterized by accuracy metrics such as confusion matrices and receiver operating characteristic (ROC) curves.

8

Conclusion and Discussion

This literature study aims to determine the specification of a Gas Source Localization (GSL) benchmark such that it is accessible, has the right balance between practicality and realism, and provides valuable performance metrics. With the variety of GSL methods ever increasing objective performance comparison becomes more and more relevant. Unfortunately, study shows these comparisons to be rare in the research field. Often, results are unable to be fairly compared due to differences in methods, namely the environment setup (arena size, presence of obstacles) and gas simulation (model fidelity, wind conditions). The development of a broadly accessible benchmark will therefore alleviate these limitations holding back the research field.

From the survey of methods arise initial GSL simulator/benchmark requirements. Most importantly, the simulator must be able to offer a robot instantaneous gas concentration data for a specific location and time. Additionally, the agent must be able to collect wind field data if applicable. Furthermore, computational efficiency is desirable. Especially with the rising popularity of multi-agent and ML methods computational efficiency becomes increasingly vital.

A diverse selection of general and GSL-specific simulators is studied to consider existing solutions. The choice of a simulator will ultimately be decided after some initial experimentation. The most widely used option is Gazebo in combination with ROS and GADEN. As Gazebo and GADEN are well established in the research field they present a good option for creating a benchmark that is widely accessible. However, Gazebo can become computationally expensive and is not photorealistic. These shortcomings hinder large-scale online Machine Learning (ML) and the simulation of vision sensors. Compared to Gazebo, Unity improves on visual fidelity as it was originally a game engine. Nevertheless, good examples such as Flightmare and Flightgoggles use Unity for rendering. With Unity, the physics is usually handled separately for computational efficiency. Finally, Unity is readily accessible due to excellent documentation and the surrounding community. Lastly, a relatively new simulator to consider is NVIDIA's Isaac Sim. The focus of this simulator lies in minimizing the simulation to reality gap by the use of photo-realistic rendering and accurate physics. For accessibility, it features APIs for ROS and Python and features an ML toolbox. Unfortunately, Isaac Sim is closed-source and relatively new, therefore adoption might pose a challenge.

Finally, types of performance metrics are evaluated. Success rate, movement overhead and search time are considered to be the most commonly used metrics and are therefore important to include in the benchmark. For methods predicting the location of the odour source or the distribution of concentration, accuracy and Earth Mover's Distance (EMD) can also be of use.

Continuing the thesis work involves experimentation with different simulation environment setups starting with NVIDIA's Isaac Sim as it shows great potential. However, if Isaac Sim is deemed too impractical the focus will shift towards Flightmare because the local research group at the TU Delft already has experience with this software. The main challenges are going to be the integration of gas plume simulation and the generation of environments. Additionally, the simulation might be validated by comparing the results from previous real-world experimentation.

Part III

Additional Results & Closure

9

AutoGDM+

AutoGDM+ (Automatic Gas Dispersion Modeling) serves as the environment generation pipeline for GSL-Bench. This includes the creation of environment scenes and the simulation of wind fields and gas dispersion, see Figure 9.1. The '+' refers to the fact that this generation pipeline is an evolution of the original AutoGDM presented by Duisterhof et al. [5]. The output of AutoGDM+ is not specifically tailored to the use in GSL-Bench and is easily integrated into any other simulation project. Therefore, the goal of this chapter is to describe the pipeline in more detail and facilitate the use of AutoGDM+ outside of this study. Please note that these concepts build on the information from the scientific article in Part I.

This chapter is structured according to the gray boxes in the workflow of Figure 9.1. Layout (or scene) generation is elaborated upon in Section 9.1. Subsequently, the wind field simulation and its structure is formulated in Section 9.2. Finally, the gas dispersion simulation is explained in Section 9.3

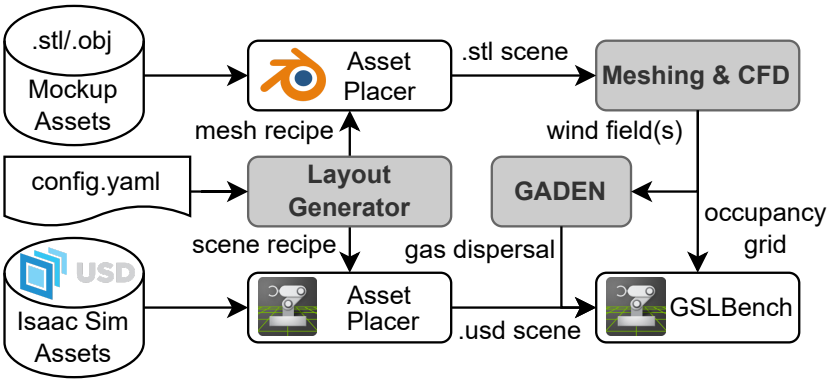


Figure 9.1: Workflow of AutoGDM+. The respective asset placers create a scene from a generated layout. The .stl scene is used for meshing and CFD to generate wind fields and ultimately gas dispersion data. The .usd scene is used directly with GSL-Bench in Isaac Sim.

9.1. Layout Generator

The layout generator is the starting point of the generation pipeline. It allows for automatic variation between multiple generated environments of the same type, making it useful for Machine Learning (ML) applications. Variation applies to the type of asset (empty racks versus full racks), their location and orientation. The layout generator outputs a recipe `.txt` file from the relevant settings in the `conf.yaml` that is passed to the asset placers. Although Figure 9.1 shows a 'mesh recipe' and a separate 'scene recipe', they are contained in this single recipe file. The top-level settings of the layout generator in the `conf.yaml` are described in Table 9.1. The remainder of this section elaborates on the available environment types, the generated recipes and the respective asset placers.

9.1.1. Environment Types

Currently, AutoGDM+ features three warehouse environment types; the empty, simple and complex warehouse respectively. The boundary conditions regarding the wind generation are largely identical

Table 9.1: Layout generator parameters

Setting	Example Value	Description
`env_types`	`['wh_empty', 'wh_simple']`	list of the available environment types
`env_type`	`1`	selected environment type (`'wh_simple'`)
`env_amount`	`20`	amount of environments to be generated
`env_size`	`[15.0, 15.0, 8.0]`	environment size in XYZ (m)

across all warehouse types (as described by the scientific paper) and are configurable with the parameters in Table 9.2. The location of the inlet and outlet are chosen such to promote significant airflow through the entire scene. Walls and lights are added to the scene for all warehouse types. By default, Isaac Sim features walls that have a white base of $3\,\mathrm{m}$ and continue with bright yellow walls. Arguably, an all-white wall is more realistic and can be specified with the configuration file. Figure 9.2 illustrates the difference in wall types. The lights are placed just above the specified environment height in order to provide lighting for the entire scene.

Table 9.2: Warehouse environment generation parameters

Setting	Example Value	Description
`inlet_size`	`[1.5, 2.4]`	inlet size in YZ (m)
`inlet_vel`	`[1.0, 0.0, 0.0]`	inlet velocity in XYZ ($\mathrm{m\,s^{-1}}$)
`outlet_size`	`[1.5, 2.4]`	outlet size in YZ (m)
`emptyfullrackdiv`	`0.2`	least percentage of filled racks
`white_walls`	`True`	choice between all-white or white-yellow walls

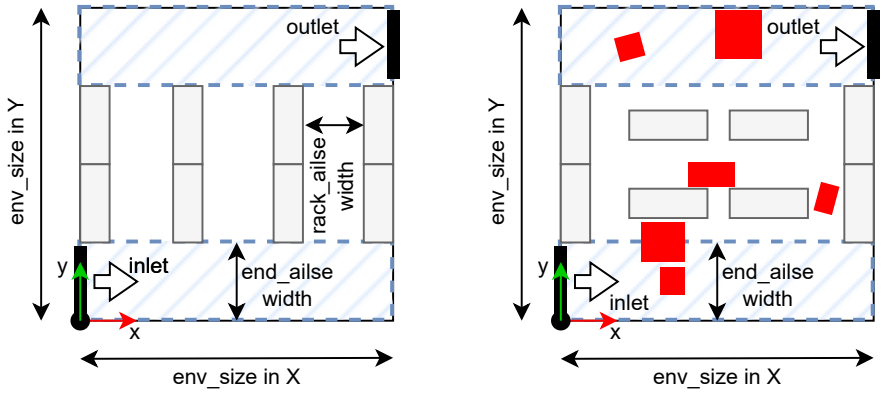
(a) Empty warehouse with the default white-yellow walls **(b)** Complex warehouse with the all-white walls

Figure 9.2: Different wall options for the warehouse environment

The simple warehouse environment introduces racks to the scene. These racks are placed in rows such that they leave at least the required space for the end aisles and rack aisles, see Figure 9.3. The complex warehouse layout includes extra obstacles rotated at a random angle as depicted in red. These obstacles include forklifts and piles of warehouse items. Additionally, the rows of racks that are not against the wall are turned $90°$ to create tighter, more challenging passages.

9.1.2. Recipe Dictionary

Figure 9.4 gives an overview of the contents of a recipe. The recipe text file is mainly a dictionary of nine lists of asset dictionaries. It also contains entries for the environment type and size in combination with the inlet and outlet sizes. The nine lists of assets are used for three purposes: CFD, Gas dispersion and Isaac Sim scene generation. Only the list of interior assets is used for all three purposes, the other lists are created separately due to the differences in requirements of the subsequent steps. The methods that create these lists of assets are contained within the warehouse environment class. This class in initialized



(a) Simple warehouse layout with the end-aisles marked by the hatched boxes

(b) Complex warehouse layout with additional obstacles in red

Figure 9.3: Schematic layout for the simple and complex warehouse environments.

with assets used throughout the class for convenience. The assets themselves are instances of the asset class as well. Most importantly, the asset class contains a method that returns a dictionary of its instance attributes.

The lists for the purpose of CFD and gas dispersion are used by the Blender asset placer. The blender asset placer is a script that uses the Blender Python API to place the assets and export the multiple required meshes. The list of sides, outlets and inlets contain definitions of planes that define the boundaries for CFD in space. It is important to note that the normal vectors of these planes must point inwards for the subsequent meshing process.

The list of interior assets is unique because of the included 'mockups'. A mockup is a low resolution mesh of the associated model that still contains the relevant geometry. The high fidelity models that are bundled with Isaac Sim are considered to be too detailed for CFD. For example, the file size of an interior mesh for a simple warehouse of 20x30x8m is reduced from 750MB to just 86kB. The asset class accommodates the mockup meshes by defining the filename as a list of two paths; one to the original Isaac Sim asset, and the other to the corresponding mockup asset. The mockup assets are easily created manually in Blender.

The gas dispersion simulation also makes use of the low resolution interior assets in order to create an occupancy grid. Additionally, there are two lists created solely for the dispersion simulation because of the required geometry. In contrast to the models for CFD, the gas dispersion simulation requires the inlets, outlets and walls with thickness. All of the geometry is modeled with cubes in Blender. Boolean operations are used to create the walls by subtracting the inside, inlet and outlet from the wall geometry. These operations are specified in the `gaden_bools` list. By default, the wall thickness is set to 0.2 m for the warehouse environments.

Finally, assets purely for Isaac Sim contain the floor tiles, walls and lights. These are all models provided with Isaac Sim. The floor tiles are 6x6m and are tiled to cover the entire scene. The walls consists of straight walls and corner pieces. The model of the light fixture is the only Isaac Sim asset that is changed from its original state. The original fixture did not emit light because it was turned off, whereas the custom asset features the light fixture turned on.

9.2. Wind Data Generation

The wind generation pipeline consists of a series of functions that automate the use of OpenFOAM. Note that AutoGDM+ uses OpenFOAM *ESI* `v2212` (instead of Foundation) specifically because it already includes the preferred method for meshing. The environments are solved one by one after they are meshed. Meshing settings are verified with a mesh sensitivity analysis presented in this section. If the CFD fails for an environment it is disregarded for the subsequent steps in order to complete the remaining environments.

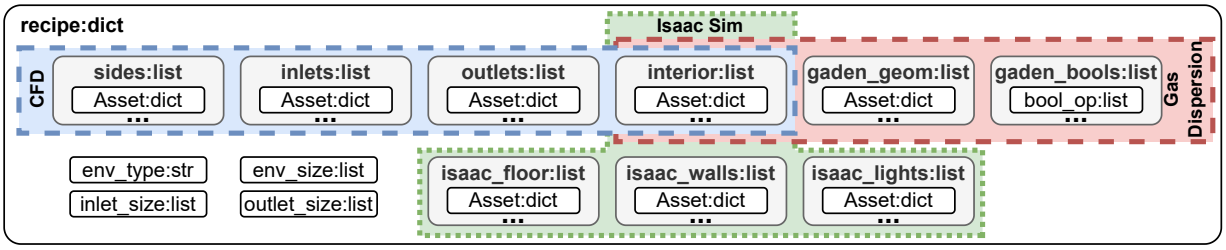
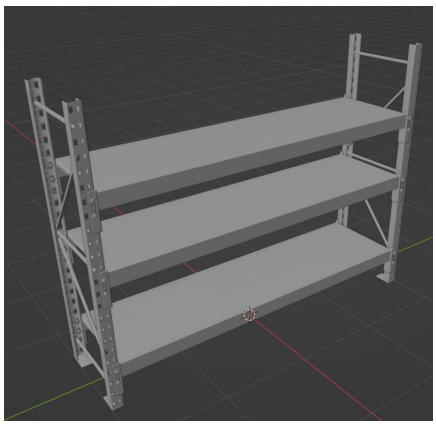
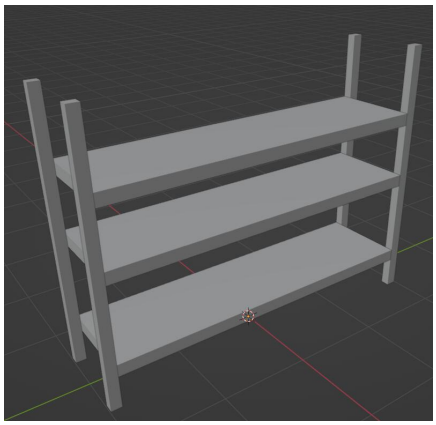


Figure 9.4: Contents of the recipe text file that is created by the layout generator. In general, a recipe is a dictionary of lists containing dictionaries of assets. The list of interior assets is used for all purposes.



(a) High resolution mesh of a shelve included with Isaac Sim consisting of 27348 vertices.



(b) Manually replicated mockup of the shelve consisting of only 56 vertices.

Figure 9.5: High resolution mesh of a shelve **(a)** and its corresponding mockup model **(b)**. The mockup models are used for the CFD and gas dispersion process.

9.2.1. Meshing

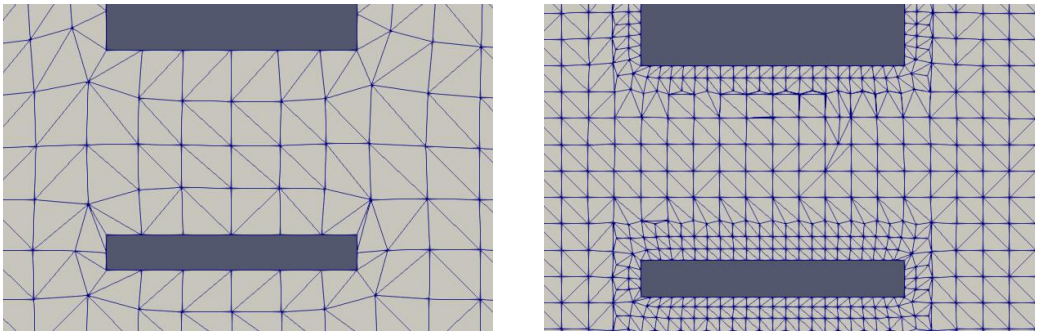
Meshing is performed with OpenFOAM's `cfMesh`. `snappyHexMesh` was also considered and tested as it is used by the original AutoGDM. Compared to `cfMesh`, `snappyHexMesh` proved to be less robust and presents a more complicated workflow. Overall, `cfMesh` offers sufficient mesh quality and is therefore the meshing method of choice. The available meshing parameters for AutoGDM+ are presented by Table 9.3. In general, robust meshing performance is observed with uniform meshing parameters and no `localRefinement`. If specified, AutoGDM+ is set to apply the `localRefinement` around the interior assets. Additionally, a uniform cell size keeps the mesh file size and computation time manageable. Luckily, `cfMesh` is a hyperthreaded process by default for a significant decrease in computation time. Figure 9.6 illustrates the effect of `localRefinement` with a cross section of the mesh.

Table 9.3: Meshing parameters. These example parameters provide consistent mesh results.

Setting	Example Value	Description
`minCellSize`	`0.2`	activates automatic refinement of the mesh (m)
`maxCellSize`	`0.2`	default cell size used for the meshing job (m)
`boundaryCellSize`	`0.2`	refinement of cells at the boundaries of the environment (m)
`localRefinement`	`0.0`	refinement around interior assets (m) (`0.0`==no refinement)

9.2.2. Mesh Sensitivity Analysis

A mesh sensitivity analysis is performed to determine an appropriate mesh resolution. A simple warehouse environment with a size of 10x16x8m is meshed with the settings shown in Table 9.4. The inlet and outlet have a size of 1.5 m wide by 2.4 m tall, and the inlet velocity is $1\,\mathrm{m\,s^{-1}}$ in the positive Y direction. The



(a) Mesh created with the settings from Table 9.3

(b) Mesh created with the min, max and boundary set to 0.1 instead of 0.2 and a `localRefinement` of 0.05

Figure 9.6: Cross section of meshes with different settings that slice through warehouse shelves to highlight the effect of a `localRefinement`.

simulation is performed using the PISO algorithm with a variable time step and a maximum Courant number of 1 (see Section 9.2.3 for more details). The magnitude of the wind vector is the parameter of interest and is probed at 4 different locations at a time of 50 s.

The results show that the wind vector is fairly consistent across different environments at location 3 and 4. However, the results from location 1 and 2 show display a significant amount of variation. Because these locations are closer to the inlet they experience larger fluctuations in wind velocity which can contribute to this variation. Unfortunately, the magnitudes do not seem to approach a certain value, rendering the results largely inconclusive. Nevertheless, a uniform cell size of 0.2 m is deemed to be the best compromise between spacial resolution and computational times for now.

Table 9.4: Mesh sensitivity analysis parameters. *The gas dispersion simulation expects a uniform cell size, but also works with meshes containing some local refinement.

Environment	**1***	**2**	**3**	**4**	**5**
`minCellSize, maxCellSize, boundaryCellSize`	`0.5`	`0.4`	`0.25`	`0.2`	`0.1`
`localRefinement`	`0.1`	`0.0`	`0.0`	`0.0`	`0.0`

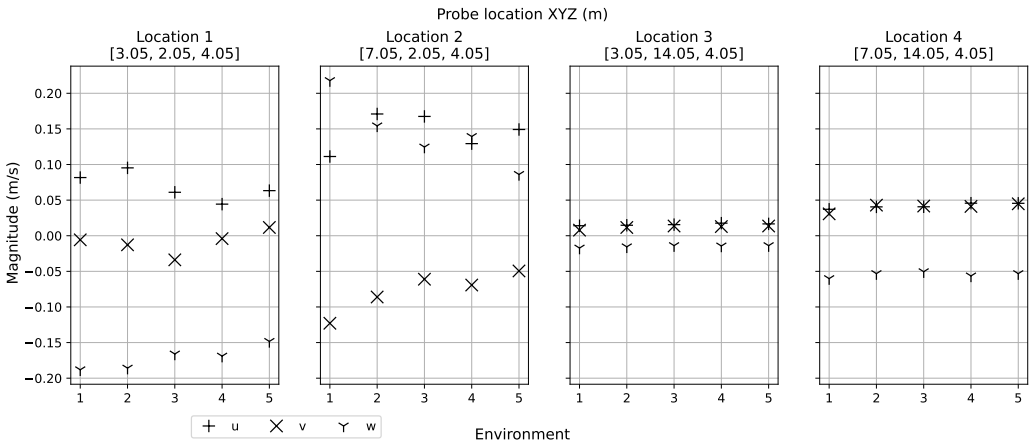


Figure 9.7: Mesh sensitivity analysis results of four different locations from the environments described by Table 9.4. The results are consistent at locations 3 and 4. However, locations 1 and 2 show more variation in their results, probably due to the fact that they are closer to the inlet.

9.2.3. CFD Simulation

Currently, `pimpleFoam` is the solver used by AutoGDM+. It is a large time-step transient solver for incompressible, turbulent flow. The 'pimple' solver combines the PISO and SIMPLE algorithms making it comparably versatile. The result is a merger of the transient simulation of the PISO algorithm and residual control of the SIMPLE algorithm, allowing for transient simulations over larger timescales that involve a Courant number > 1. By default however, pimpleFoam runs in PISO mode because the `nOuterCorrectors` is set to 1. By limiting the number of outer correctors, the amount of calculation is reduced while retaining accurate results. The simulation can be sped up by simulating at larger timescales and higher Courant numbers when accuracy is of a lesser importance. This is done by setting the `maxCo` to a desired value and increasing the number of outer correctors to 10 to prevent the simulation from diverging. If the simulation is unstable the number of outer correctors can be increased, or `maxCo` can be decreased, or the mesh cell size can be increased.

Table 9.5 shows all the configurable parameters that apply to the CFD simulation. It is observed that the simulation is the quickest when all but two (virtual) threads are dedicated to the simulation. This is the default setting for AutoGDM+, but any other number of threads can be assigned to the simulation with the `threads` setting.

Table 9.5: CFD Solving parameters

Setting	Example Value	Description
`threads`	`0`	set to `0` to use all but 2 threads for maximum performance.
`endTime`	`5.0`	end time of the wind simulation (s)
`writeInterval`	`1.0`	time interval at which the results are saved (s)
`maxCo`	`1.0`	maximum allowed Courant number
`maxDeltaT`	`0.0`	maximum allowed time step (m) (`0.0`== inactive)
`nOuterCorrectors`	`1`	number of outer correctors in the loop (`1`== PISO mode)
`latestTime`	`True`	use the latest time step as steady state
`timeRange`	`3:5`	select a time range of wind fields (`latestTime` must be `False`)

9.2.4. Wind Data Post-Processing

Post processing the CFD simulation data starts with exporting the wind vector 'U' at the cell centers. The `timeRange` parameter defines the wind fields that are exported, but if `latestTime` is set to true, only the last wind field is exported for further processing. This data is converted to a `.csv` file so that it can be used by GADEN, the gas dispersion simulation. With the environment size and cell size known, GADEN converts the `.csv` file to three, one-dimensional vectors that contain the X, Y and Z components of all the wind vectors in the wind field. Spreading out this data into a one dimensional vector is done for performance reasons. Finally, all iterations of the wind vector (in the case of multiple wind fields) are converted from binary and combined into a numpy array.

The desired index of this one-dimensional wind vector is accessed by first determining the cell indices as shown by Equation 9.1, with $\mathbf{x}_{loc}$ the location in XYZ, $\mathbf{x}_{min}$ the scene minimums and $\Delta\mathbf{x}$ the cell size in XYZ. The cell indices are then used in Equation 9.2 to determine the right index of the wind vector.

$$\mathbf{i} = \left\lceil \frac{\mathbf{x}_{loc} - \mathbf{x}_{min}}{\Delta\mathbf{x}} \right\rceil \tag{9.1}$$

$$i_{wind} = i_x + i_y n_{cells_x} + i_z n_{cells_x} n_{cells_y} \tag{9.2}$$

9.3. Gas Data Generation

The final step of the pipeline features the gas dispersion simulation with the affection-diffusion method discussed in Part II. It requires three inputs; the settings displayed in Table 9.6, the `.stl` mesh geometry created by the Blender asset placer and the the post-processed wind field as described earlier. The settings specify the placement of the gas source and the type of gas. These gases differ in their specific

gravity, making them either sink to the ground or rise up (if left in undisturbed air). They also influence the response of the simulated gas sensors. `sim_time` and `time_step` determine the run time and time resolution of the simulation respectively. The time resolution should be set to a value of approximately the cell size divided by the maximum wind speed. The last three settings allow for looping through multiple wind fields during the gas dispersion simulation. For example, with the settings in Table 9.5 there would be three wind fields from timesteps 3, 4, and 5 seconds respectively. With these settings, the gas simulation will loop through these three available wind fields. Note that `wind_looping` can still be true even if the CFD setting `latestTime` is also true, because GADEN then automatically loops the single present windfield.

Table 9.6: Gas dispersion simulation parameters

Setting	Example Value	Description
`src_placement_types`	`['specific', 'random']`	list of gas source placement types
`src_placement_type`	`0`	selection of the gas source placement type
`src_placement`	`[5.0, 1.0, 2.0]`	source placement in XYZ (m)
`gas_type`	`0`	`0`= ethanol, `1`= methane, `2`= hydrogen
`sim_time`	`500`	gas dispersion simulation time (s)
`time_step`	`0.2`	time resolution of the simulation (s)
`wind_looping`	`'true'`	loop though multiple wind fields
`wind_start_step`	`0`	first wind iteration for looping
`wind_stop_step`	`3`	last wind iteration for looping

Like the wind data, the resulting gas data is converted from binary to numpy arrrays for ease of use. Currently, each numpy array contains one iteration of the gas simulation with the location and size of all the filaments present. Additionally, the header information containing variables such as the amount of moles per filament is also converted.

10

GSL-Bench

GSL-Bench refers to the robotcs simulation with benchmarking capabilities together with the plotting of metrics. As shown by Figure 10.1, it utilizes Pegasus Simulator [29], an Isaac Sim framework for aerial vehicles. The article featured in Part I concisely covers the main aspects of GSL-Bench such as the included sensors, waypoint logic and obstacle avoidance. This chapter continues by introducing two additional algorithms and their results, details on motion planning and instructions on the usage of GSL-Bench.

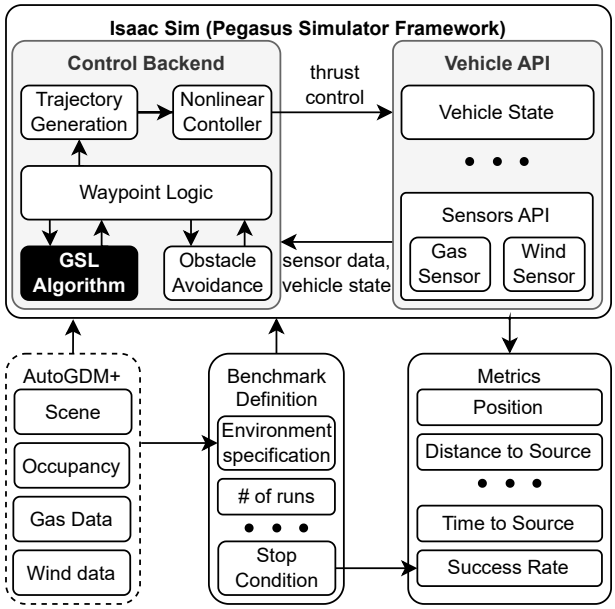


Figure 10.1: Diagram of the GSL-Bench simulation framework. The environments generated by AutoGDM+ are optional because GSL-Bench comes with six pre-generated environments.

10.1. Additional Algorithms

Two additional algorithms are implemented to demonstrate the capabilities of GSL-Bench; 3D E. coli and Sniffybug. These methods are created to showcase three dimensional and multi-agent methods respectively. Because of the key differences between all methods (including the ones presented in the article), categories are introduced to GSL-Bench so that results can be filtered and compared accordingly, see Table 10.1

10.1.1. 3D E. Coli

3D E. Coli is a variation on the E. coli method covered in the article. Instead of operating at the specified search height, it includes a z-component in its movements. The maximum magnitude of this z-component is defined by a separate surge distance `surge_dist_z`. To accommodate the added change in altitude, the surge heading is expanded from only a heading to a list containing the heading and the altitude change.

Table 10.1: GSL method categories and possible options

Dimensionality	Obstacle Avoidance	Robot Configuration	Source Declaration
2D	GSL-Bench	Single Agent	None
3D	Custom	Multi-Agent	Location
			Probability

Currently, there is no 3D obstacle avoidance to keep the algorithm from flying into obstacles. Therefore the algorithm contains its own rudimentary 'obstacle avoidance' for now. With the provided occupancy grid, it checks every goal waypoint for obstacles. The surge heading is randomized again in case of any obstruction to the goal waypoint, even if it was surging due to an increase in sensor reading, see Figure 10.2.

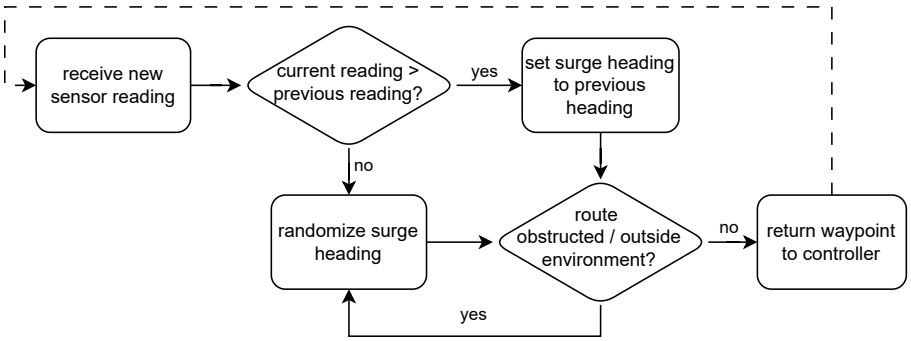


Figure 10.2: Flow diagram of the (3D) E. coli algorithm. The dashed arrow indicates omitted steps outside the scope of this method.

10.1.2. Sniffybug

The Sniffybug algorithm presented in this work is an adaptation of the original method presented by Duisterhof et al. [5]. Sniffybug is a PSO method (as outlined by algorithm 1) with a preceding exploration phase and repulsion swarming. The original Sniffybug algorithm utilizes range sensors for obstacle avoidance. These sensor are not yet available within GSL-Bench and therefore all aspects regarding 'on-board' obstacle avoidance are omitted and left to the global obstacle avoidance offered by GSL-Bench. The algorithm is also adapted to work with the 'goto waypoint, hold' cycle of GSL-Bench's included controller. This does take away some of the effectiveness of this method and will have to be improved in the future as discussed in Chapter 12.

The exploration phase has all agents traversing the environment randomly, until one of the agents senses the gas. From there, the agents update their velocity according to the PSO algorithm. When they get within a distance of `d_swarm` of each other, they initiate repulsion swarming to avoid collision. During testing, the swarm's best known position is passed to the distance to source stop condition.

10.2. Motion Planning

The motion planning included with GSL-Bench is designed to simplify the implementation any new GSL-method. The motion planning consists of a waypoint module which finds itself in the middle of the GSL-method, the obstacle avoidance and the trajectory generation as shown in Figure 10.1. How these modules work in practice is better visualized with the flow diagram depicted in Figure 10.3. In the case of aerial vehicles, the waypoint module is initialized with a take off mission consisting of an initial waypoint and one at the specified starting height. After completion of the take off mission, waypoints provided by the GSL method are used. Every path the GSL-method wants to take is checked by the obstacle avoidance module. If there are any obstructions, the obstacle avoidance module passes a set of waypoints as a mission to the waypoint module and so on.

The gray boxes in Figure 10.3 contain functions performed by the the trajectory generation module and controller. The trajectory generation module is required because the implementation of the controller

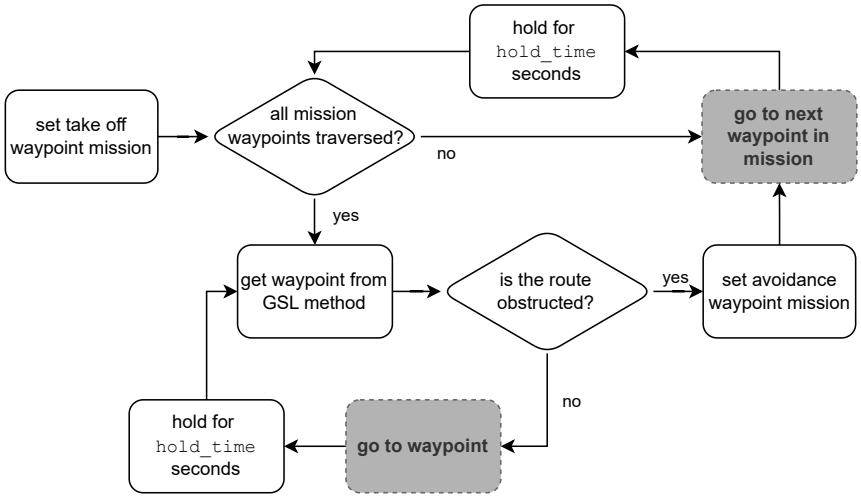


Figure 10.3: Flow diagram of the motion planning, the gray boxes contain the trajectory generation.

expects references for position, velocity, acceleration and jerk for every time step. Currently, a trajectory with minimal jerk is implemented to encourage stable flight and is generated by fitting a quintic polynomial to the start and end location, given the starting and final velocity and acceleration respectively, see Equation 10.1. An example of the generated references is shown by Figure 10.5. Generally, the velocity and acceleration at the start and end of the trajectory are both set to zero, but they could be set to different values if to traverse an environment without interruptions (if the hold time is also set to $0\,\mathrm{s}$).

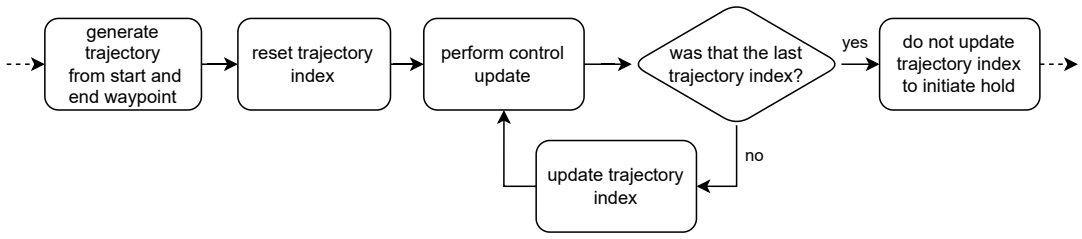


Figure 10.4: Trajectory generation and traversal loops performed in the gray boxes in Figure 10.3

$$s(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{10.1}$$

10.3. Benchmark Execution

Execution of a benchmark is facilitated by an included python script that automates the process. This section describes this process more in detail and elaborates on the simulation performance.

10.3.1. Methodology & Usage

Isaac Sim offers multiple development workflows to run your simulation[1]. This work uses the VSCode and Python workflow. Pegasus Simulator is installed as an extension as described by their documentation[2], and the simulation itself is run as a standalone Python script. The benchmarking module simply automates the execution of the standalone Python scripts. Multiple variables are used to define a benchmark: the GSL method (script), the environments and starting positions. From these inputs a complete list of experiments is generated to execute every script with every environment from every specified location. By default, each experiment is repeated ten times. Afterwards, the benchmark script may execute multiple plotting functions to visualize the requested metrics.

[1] https://docs.omniverse.nvidia.com/isaacsim/latest/index.html

[2] https://pegasussimulator.github.io/PegasusSimulator/source/setup/installation.html#installing-the-pegasus-simulator

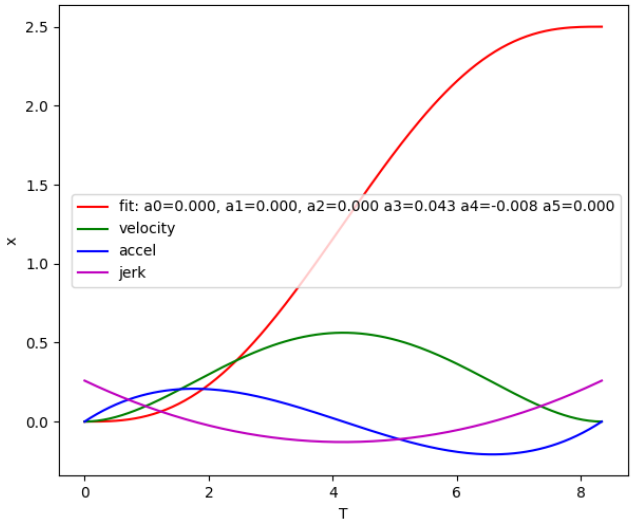


Figure 10.5: Position (fit), velocity, acceleration and jerk references over time of a minimal jerk trajectory in one dimension from 0 m to 2.5 m m generated by fitting a quintic polynomial.

10.3.2. Simulation Performance

The biggest impact on simulation performance is attributed to the sensors that are added to the Pegasus Simulator framework. Nevertheless, additional computational overhead due to the gas sensor and anemometer can be considered negligible compared to the other sensors used by the multirotor. Figure 10.6 shows a visualization of Python's `cProfile` module as a result of profiling the simulation. The Inertial Measurement Unit (IMU), GPS and barometer require orders of magnitude more computation time than the added sensors highlighted by the red rectangle. When running in headless mode, the simulation runs at approximately 1.95 real time factor.

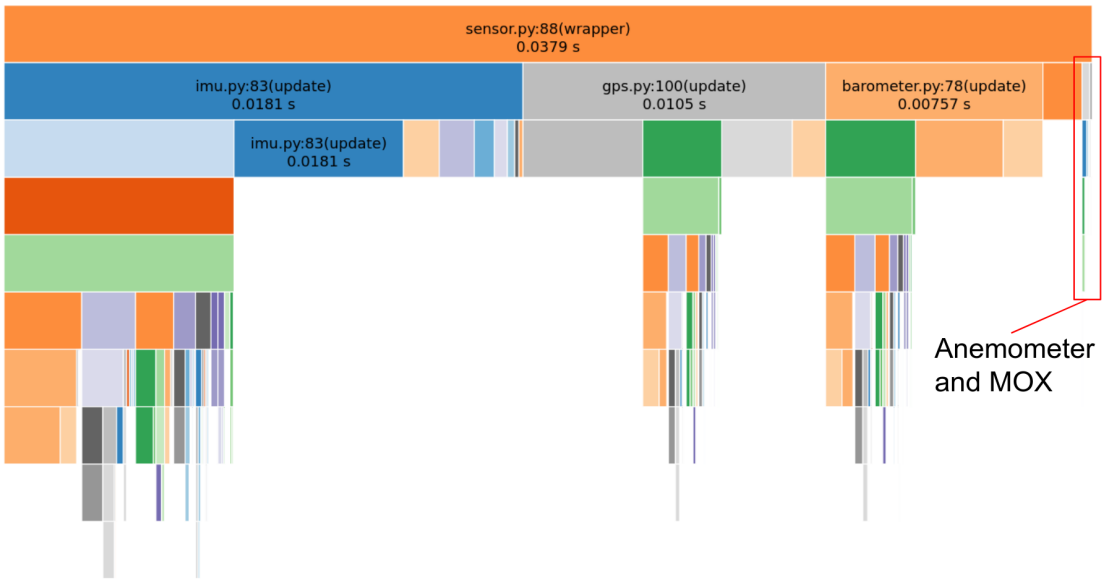


Figure 10.6: A visualization of Python's `cProfile` module showing the negligible impact the gas sensor and anemometer on the computation time (highlighted by the red rectangle to the right of `barometer.py`)

10.4. Additional Metrics

Although most of the metrics available from GSL-Bench are included in the scientific article, the visualization of the instantaneous gas reading and sensor reading are not shown. An example of this plot is depicted by Figure 10.7. In this figure, the gas concentration and sensor readings of ten runs of the dung beetle algorithm are shown. It clearly visualizes the rise and decay characteristics of the metal oxide sensor, resulting in the necessary delay compared to the instantaneous reading. Most importantly, it provides an indication of the plume tracking performance of an algorithm.

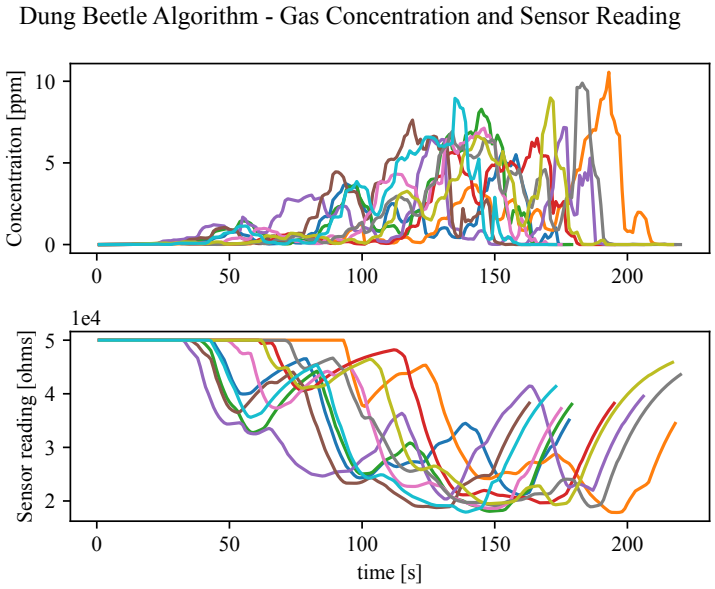


Figure 10.7: Instantaneous gas concentration and MOX sensor response over time from a dung beetle algorithm, note the rise and decay of the simulated sensor.

10.5. Additional Algorithm Results

Results from the additional algorithms discussed in Section 10.1; the 3D E. coli and adapted Sniffybug method, are presented in this section.

10.5.1. 3D E. Coli

The testing methodology of the 3D E. coli algorithm is identical to the methodology presented in the scientific article. The method is run 10 times from a grid of 9 starting positions for all 6 environments. The success rate and average time to source of all tested algorithms are shown in Figure 10.8 and Figure 10.9 respectively.

The 3D E. coli algorithm has a poor success rate compared to the 2D algorithms. There are several possible explanations for these results. Extending the search space to three dimensions reduces the chance of initial contact with the plume, thereby reducing the success rate. On the other hand, the combination of algorithm parameters such as the `surge_dist` (horizontal surge distance) and `surge_dist_z` may also have contributed to poor performance.

Whenever the 3D algorithm does find the source, it generally outperforms its 2D counterpart in environments that feature obstacles (environment 3 and higher). This can imply that the agent may be guided to the source in some way because of the current implementation of the obstacle avoidance. The longer times to the source in environment 1 and 2 reinforce this hypothesis, but may also indicate poor parameter choice. More experimentation will be required to better understand the results.

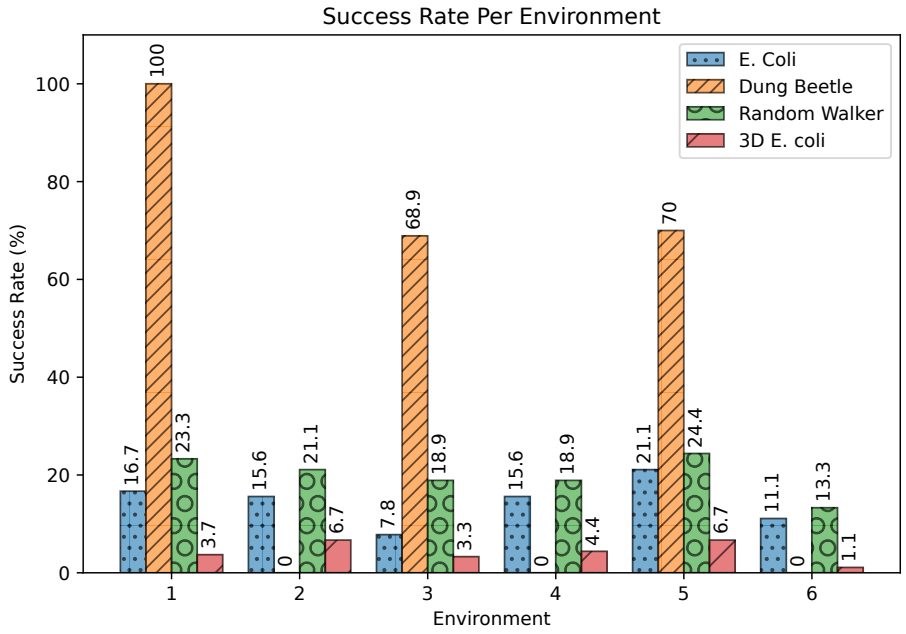


Figure 10.8: Success rate per environment for every single-agent method tested, including the addition of the 3D E. coli algorithm. 3D E. coli has the lowest success rate overall, even lower than its 2D variant. This might be explained by the extra dimension adding a too large search space.

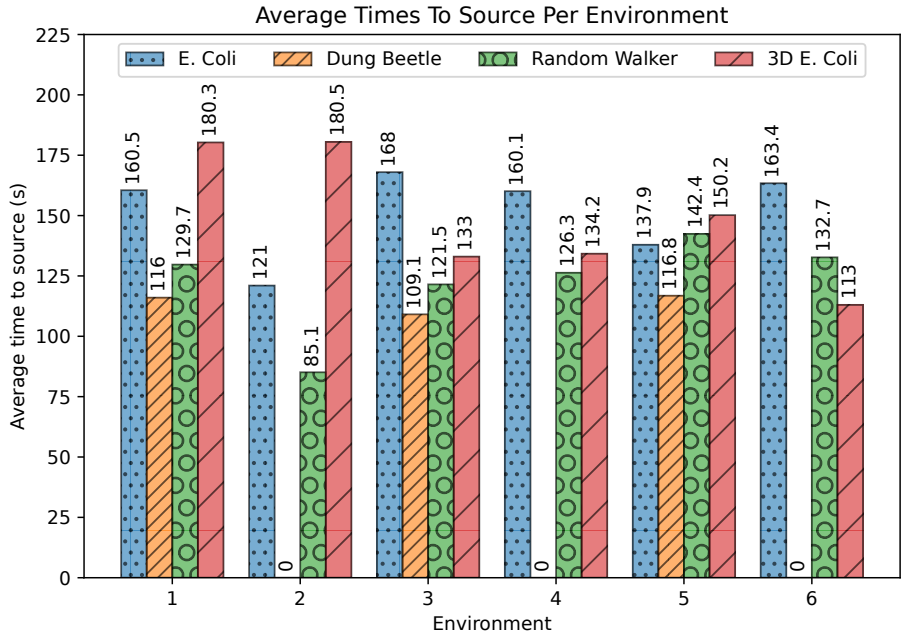


Figure 10.9: Average time to source for every single-agent method tested, including the addition of the 3D E. coli algorithm. Interestingly, the average time to source of the 3D E. coli algorithm is generally shorter in environments with obstacles $(3, 4, 5, 6)$.

10.5.2. Sniffybug

The Sniffybug algorithm was tested in environment 1 and 2 with three agents and started from the grid spaces on the diagonal as shown by Figure 10.10. The exploration phase is clearly depicted by the random movement of the agents. When one of the agents senses the gas plume, the seeking phase is initiated. This is phase is characterized by the smoother tracks. At one point, the agent starting from XY=7.5,7.5 sensed higher concentrations than the agent starting at XY=3.0,3.0, resulting in a sudden change of direction to the source location.

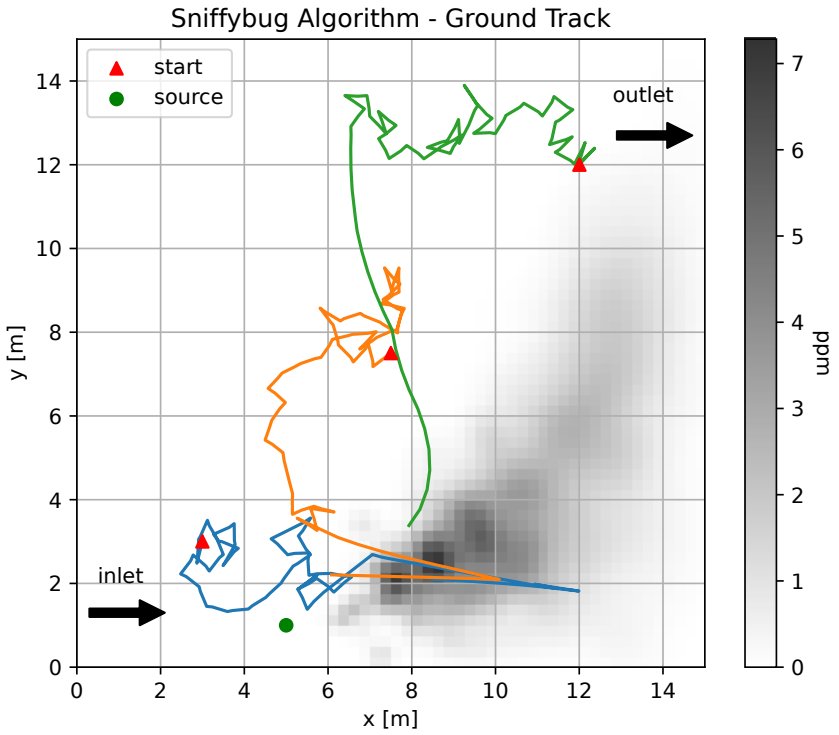


Figure 10.10: Ground tracks of the Sniffibug algorithm with agents starting at XY = 3,3; 7.5,7.5; 12,12 in environment 1. The exploration and seeking phases of the method are clearly shown in this plot.

11

Conclusion

The lack of a widely adopted benchmark for Gas Source Localization (GSL) methods slows progress in this challenging field of robotics. Current options do not offer a combination of photo-realism, high fidelity gas dispersion simulation and benchmarking functionality.

This thesis presents GSL-Bench: a high fidelity gas source localization benchmarking suite. The requirements for GSL-Bench are shaped by means of a study of literature. This study has led to a focus on accessibility and high simulation fidelity to promote widespread adoption. GSL-Bench is implemented with NVIDIA's® Isaac Sim and the Pegasus Simulator framework [29] to meet these requirements. Due to the built-in obstacle avoidance, waypoint logic and sensors one can directly concentrate their efforts on only the GSL algorithm. However, the built-in modules can easily be omitted to keep development flexible. High fidelity of the simulation is ensured with proper visuals and the introduction of AutoGDM+: a fully automated environment generation pipeline relying on validated software such as GADEN for gas dispersion simulation and OpenFOAM for CFD.

Multiple GSL algorithms are tested in six warehouse environments with increasing complexity. The results demonstrate GSL-Bench's ability to thoroughly test and visualize algorithm performance. Moreover, the performance of algorithms align with other experiments from literature, further vefifying the benchmarking suite. The GSL-Bench website[1] will provide the most recent findings and host a leaderboard showcasing results from diverse standardized test. The website will also contain instructions on how to contribute and/or test one's own methods.

[1] https://sites.google.com/view/gslbench/

12

Recommendations

Chapter 11 confirms that this work has completed the research objective stated in Section 1.1. Although the current state of the project serves as a proper basis to test GSL algorithms there are some features that are not included yet due to time constraints. This chapter provides recommendations for future work on GSL-Bench and AutoGDM+. Because the versatility of this work allows expansion into other types of research, some final recommendation to broaden the scope are given as well.

12.1. Recommendations for Future Work

In future work, development of GSL-Bench and AutoGDM+ can be continued in the following ways.

Additional controller types
A non-linear controller is included with GSL-Bench. Built around this controller are the trajectory generation and waypoint logic modules. Generally, the implementation the controller in combination with the modules serves its purpose well. However, there are cases where this kind of implementation is suboptimal. Some algorithms like the Sniffybug algorithm (discussed in Chapter 10) rely on their agents having a constant velocity to prevent deadlock. This is not yet possible with the current implementation of the controller. Therefore, additional controller types or trajectory generation modules could address these limitations.

3D obstacle avoidance
The build-in obstacle avoidance module is currently implemented in two dimensions only. The 3D E. coli algorithm therefore employs only a rudimentary obstacle check before every move. The A^* algorithm can be expanded into the third dimension to enable 3D path planning.

Additional robot types
The Pegasus Simulator framework comes bundled with a multirotor vehicle. However, the majority of research features ground-based robots such as rovers. Luckily, the Pegasus Simulator framework and Isaac Sim already support these kinds of robots, but they still have to be implemented.

Reduction in execution time
With GSL-Bench running in headless mode the simulation runs with a real-time factor of approximately 1.95. A speed increase might be achieved by making improvements in the sensor suite. For example, the gas sensor could work more efficiently if it were to work on a cell basis rather than a filament basis. This goes hand in hand with the next recommendation:

Post-Processing of the gas dispersion data
The filament data generated by AutoGDM+ is only converted from the binary to numpy arrays, each containing a list of filaments. The gas sensor has to check for each filament if it is within the limit distance for each iteration. With increasing amount of filaments in a scene the simulation noticeably slows down. This can be solved by post-processing the filament data by assigning them to cells in space as done by Odeja et al. in [26]. Or the gas concentration can already be calculated for different cells in space, like is done with the wind data to omit the filament calculation completely. This will reduce the spacial resolution however.

More extensive mesh sensitivity analysis
The mesh sensitivity analysis in Section 9.2 is largely inconclusive because the wind velocity data does not seem to converge to a certain value as expected. This could be explained by the large steps in cell count, because a reduction in cell size of 50% increases the amount of cells by a factor 8. A trend in the wind data may reveal itself with more data from more incremental steps in cell count.

Additional environment types
AutoGDM+ is currently capable of generating three types of indoor warehouse environments. Firstly, indoor environments types such as offices or homes would be a great addition. Moreover, the ability to generate non-rectangular indoor environments would increase their complexity. Besides the indoor environments, outdoor environments also play a large part in GSL research.

12.2. Recommendations to Broaden the Scope

Although GSL-Bench and AutoGDM+ are currently developed for their intended purpose, they might prove useful for other applications as well. The following uses would require little adaptation to the presented work:

Machine Learning Environment
Isaac Sim provides an extension for reinforcement learning training and inference: Isaac Gym[1]. This way, training RL agents can be run in parallel for a significant decrease in computation time.

Manipulation and end effector research
Manipulation is often simulated with Isaac Sim. Because the physics are accelerated using the GPU they can be of high fidelity. Combining UAV's with manipulation in Isaac Sim therefore require little alterations to the simulation of GSL-Bench.

Wind-only generation
Other research might only be interested in the wind data produced by AutoGDM+. Currently, the gas dispersion simulation is required to generate the wind data. Because using the gas dispersion simulation requires extra steps during the installation process and runtime, a version of AutoGDM+ that only generates the wind data without the gas dispersion simulation would be a welcome addition.

[1] `https://docs.omniverse.nvidia.com/isaacsim/latest/tutorial_gym_isaac_gym.html`

References

[1] B. Lorena Villarreal et al. “Synthesis of odor tracking algorithms with genetic programming”. In: *Neurocomputing* 175 (2016), pp. 1019–1032. DOI: 10.1016/j.neucom.2015.09.108. URL: http://dx.doi.org/10.1016/j.neucom.2015.09.108.

[2] Chiara Ercolani et al. “3D odor source localization using a micro aerial vehicle: System design and performance evaluation”. In: *IEEE International Conference on Intelligent Robots and Systems* (2020), pp. 6194–6200. DOI: 10.1109/IROS45743.2020.9341501.

[3] Shuo Pang et al. “Chemical plume source localization”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 36.5 (2006), pp. 1068–1080. DOI: 10.1109/TSMCB.2006.874689.

[4] C. D. Jones. “On the structure of instantaneous plumes in the atmosphere”. In: *Journal of Hazardous Materials* 7.2 (1983), pp. 87–112. DOI: 10.1016/0304-3894(83)80001-6.

[5] Bardienus P. Duisterhof et al. “Sniffy Bug: A Fully Autonomous Swarm of Gas-Seeking Nano Quadcopters in Cluttered Environments”. In: *IEEE International Conference on Intelligent Robots and Systems* (2021), pp. 9099–9106. DOI: 10.1109/IROS51168.2021.9636217.

[6] H. Ishida et al. *Odour-source localization system mimicking behaviour of silkworm moth*. 1995. DOI: 10.1016/0924-4247(95)01220-6.

[7] R. Andrew Russell et al. “A comparison of reactive robot chemotaxis algorithms”. In: *Robotics and Autonomous Systems* 45.2 (2003), pp. 83–97. DOI: 10.1016/S0921-8890(03)00120-9.

[8] Wisnu Jatmiko et al. “A Mobile Robots PSO-based for Odor Source Localization in Dynamic Advection-Diffusion Environment”. In: *International Conference on Intelligent Robots and Systems* (2006), pp. 4527–4532.

[9] Xiaohui Hu et al. “Adaptive particle swarm optimization: Detection and response to dynamic systems”. In: *Proceedings of the 2002 Congress on Evolutionary Computation, CEC 2002* 2 (2002), pp. 1666–1670. DOI: 10.1109/CEC.2002.1004492.

[10] Fei Li et al. “Probability-PSO algorithm for multi-robot based odor source localization in ventilated indoor environments”. In: *Lecture Notes in Computer Science* 5314 LNAI.PART 1 (2008), pp. 1206–1215. DOI: 10.1007/978-3-540-88513-9{_}128.

[11] Joseph R. Bourne et al. “Coordinated Bayesian-Based Bioinspired Plume Source Term Estimation and Source Seeking for Mobile Robots”. In: *IEEE Transactions on Robotics* 35.4 (2019), pp. 967–986. DOI: 10.1109/TRO.2019.2912520.

[12] Yaqub A Prabowo et al. “Integration of Bayesian Inference and Anemotaxis for Robotics Gas Source Localization in a Large Cluttered Outdoor Environment”. In: *IEEE Access* PP (2023), p. 1. DOI: 10.1109/ACCESS.2023.3238470.

[13] Jay A. Farrell et al. “Plume Mapping via Hidden Markov Methods”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* 33.6 (2003), pp. 850–863. DOI: 10.1109/TSMCB.2003.810873.

[14] Jian Long Wei et al. “Multi-Robot gas-source localization based on reinforcement learning”. In: *2012 IEEE International Conference on Robotics and Biomimetics, ROBIO 2012 - Conference Digest* (2012), pp. 1440–1445. DOI: 10.1109/ROBIO.2012.6491171.

[15] G. C.H.E. de Croon et al. “Evolutionary robotics approach to odor source localization”. In: *Neurocomputing* 121 (2013), pp. 481–497. DOI: 10.1016/j.neucom.2013.05.028. URL: http://dx.doi.org/10.1016/j.neucom.2013.05.028.

[16] Cheng Song et al. “Collaborative infotaxis: Searching for a signal-emitting source based on particle filter and Gaussian fitting”. In: *Robotics and Autonomous Systems* 125 (2020), p. 103414. DOI: 10.1016/j.robot.2019.103414. URL: https://doi.org/10.1016/j.robot.2019.103414.

[17] Kumar Gaurav et al. “Single and multiple odor source localization using hybrid nature-inspired algorithm”. In: *Sadhana - Academy Proceedings in Engineering Sciences* 45.1 (2020), pp. 1–19. DOI: 10.1007/s12046-020-1318-3. URL: https://doi.org/10.1007/s12046-020-1318-3.

[18] Mohamed Awadalla et al. “3D framework combining CFD and MATLAB techniques for plume source localization research”. In: *Building and Environment* 70 (2013), pp. 10–19. DOI: 10.1016/j.buildenv.2013.07.021. URL: http://dx.doi.org/10.1016/j.buildenv.2013.07.021.

[19] Ali Marjovi et al. “Optimal spatial formation of swarm robotic gas sensors in odor plume finding”. In: *Auton Robot* 35 (2013), pp. 93–109. DOI: 10.1007/s10514-013-9336-1.

[20] Gabriele Ferri et al. “Mapping multiple gas/odor sources in an uncontrolled indoor environment using a Bayesian occupancy grid mapping based method”. In: *Robotics and Autonomous Systems* 59.11 (2011), pp. 988–1000. DOI: 10.1016/j.robot.2011.06.007. URL: http://dx.doi.org/10.1016/j.robot.2011.06.007.

[21] Victor Hernandez Bennetts et al. “Mobile robots for localizing gas emission sources on landfill sites: Is bio-inspiration the way to go?” In: *Frontiers in Neuroengineering* 4.JANUARY (2012), pp. 1–12. DOI: 10.3389/fneng.2011.00020.

[22] Faezeh Rahbar et al. “A 3-D bio-inspired odor source localization and its validation in realistic environmental conditions”. In: *IEEE International Conference on Intelligent Robots and Systems* 2017-Septe (2017), pp. 3983–3989. DOI: 10.1109/IROS.2017.8206252.

[23] Manolis Savva et al. “Habitat: A platform for embodied AI research”. In: *Proceedings of the IEEE International Conference on Computer Vision* 2019-Octob (2019), pp. 9338–9346. DOI: 10.1109/ICCV.2019.00943.

[24] Winter Guerra et al. “FlightGoggles: A Modular Framework for Photorealistic Camera, Exteroceptive Sensor, and Dynamics Simulation”. In: *International Conference on Intelligent Robots and Systems* (2019). DOI: 10.1109/IROS40897.2019.8968116. URL: http://arxiv.org/abs/1905.11377%0Ahttp://dx.doi.org/10.1109/IROS40897.2019.8968116.

[25] Javier Monroy et al. “GADEN: A 3D gas dispersion simulator for mobile robot olfaction in realistic environments”. In: *Sensors (Switzerland)* 17.7 (2017), pp. 1–16. DOI: 10.3390/s17071479.

[26] Pepe Ojeda et al. “A simulation framework for the integration of artificial olfaction into multi-sensor mobile robots”. In: *Sensors* 21.6 (2021), pp. 1–13. DOI: 10.3390/s21062041.

[27] G Cabrita et al. “PlumeSim-Player/Stage Plume Simulator”. In: *ICRA Workshop on Networked and Mobile Robot Olfaction in Natural, Dynamic Environments* (2010). URL: papers://15a17785-8386-4a79-b6ae-1c6e2d0ed658/Paper/p5861.

[28] Mario Coppola et al. “Provable self-organizing pattern formation by a swarm of robots with limited knowledge”. In: *Swarm Intelligence* 13.1 (2019), pp. 59–94. DOI: 10.1007/s11721-019-00163-0. URL: https://doi.org/10.1007/s11721-019-00163-0.

[29] Marcelo Jacinto et al. “Pegasus Simulator: An Isaac Sim Framework for Multiple Aerial Vehicles Simulation”. In: (July 2023). URL: http://arxiv.org/abs/2307.05263.

[30] Tao Jing et al. “Recent Progress and Trend of Robot Odor Source Localization”. In: *IEEJ Transactions on Electrical and Electronic Engineering* 16.7 (2021), pp. 938–953. DOI: 10.1002/tee.23364.

[31] Xin xing Chen et al. “Odor source localization algorithms on mobile robots: A review and future outlook”. In: *Robotics and Autonomous Systems* 112 (2019), pp. 123–136. DOI: 10.1016/j.robot.2018.11.014. URL: https://doi.org/10.1016/j.robot.2018.11.014.

[32] Gideon Kowadlo et al. “Robot odor localization: A taxonomy and survey”. In: *International Journal of Robotics Research* 27.8 (2008), pp. 869–894. DOI: 10.1177/0278364908095118.

[33] Roberto Rozas et al. “Artificial Smell Detection for Robotic Navigation”. In: *Advanced Robotics* 4.1 (1991), pp. 1730–1733. DOI: 10.1109/ICAR.1991.240354.

[34] Nicole Voges et al. “Reactive Searching and Infotaxis in Odor Source Localization”. In: *PLoS Computational Biology* 10.10 (2014). DOI: 10.1371/journal.pcbi.1003861.

[35] Ziqi Chen et al. “Underground Odor Source Localization Based on a Variation of Lower Organism Search Behavior”. In: *IEEE Sensors Journal* 17.18 (2017), pp. 5963–5970. DOI: 10.1109/JSEN.2017.2729558.

[36] R Kanzaki et al. “Self-Generated Zigzag Turning of Bombyx-Mori Males during Pheromone-Mediated Upwind Walking”. In: *Zoological Science* 9.3 (1992), pp. 515–527.

[37] Erol Şahin et al. “Swarm Robotics”. In: *Swarm Intelligence* (2008), pp. 87–100. DOI: 10.1007/978-3-540-74089-6{_}3.

[38] V. Genovese et al. “Self organizing behavior and swarm intelligence in a pack of mobile miniature robots in search of pollutants”. In: *IEEE International Conference on Intelligent Robots and Systems* 3 (1992), pp. 1575–1582. DOI: 10.1109/IROS.1992.594225.

[39] Yuli Zhang et al. “A Virtual Physics-based Approach to Chemical Source Localization using Mobile Robots”. In: *Applied Mechanics and Materials* 263-266 (2013), pp. 674–679. DOI: 10.4028/www.scientific.net/AMM.263-266.674.

[40] Jorge M. Soares et al. “A distributed formation-based odor source localization algorithm - Design, implementation, and wind tunnel evaluation”. In: *Proceedings - IEEE International Conference on Robotics and Automation* 2015-June.June (2015), pp. 1830–1836. DOI: 10.1109/ICRA.2015.7139436.

[41] James Kennedy et al. *Swarm Intelligence*. Morgan Kaufmann Publishers, 2001.

[42] Upma Jain et al. “Multiple odor source localization using diverse-PSO and group-based strategies in an unknown environment”. In: *Journal of Computational Science* 34 (2019), pp. 33–47. DOI: 10.1016/j.jocs.2019.04.008. URL: https://doi.org/10.1016/j.jocs.2019.04.008.

[43] James Kennedy et al. “Particle Swarm Optimization”. In: *International Conference on Neural Networks* (1995), pp. 1942–1948. DOI: 10.1007/978-3-319-46173-1{_}2.

[44] Tim Blackwell et al. “Multi-swarm optimization in dynamic environments”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3005 (2004), pp. 489–500. DOI: 10.1007/978-3-540-24653-4{_}50.

[45] Wisnu Jatmiko et al. “A PSO-based mobile robot for odor source localization in dynamic advection-diffusion with obstacles environment: theory, simulation and measurement”. In: *IEEE Computational Intelligence Magazine* (2007), pp. 37–51.

[46] K. N. Krishnanand et al. “Detection of multiple source locations using a glowworm metaphor with applications to collective robotics”. In: *Proceedings - 2005 IEEE Swarm Intelligence Symposium, SIS 2005* 2005 (2005), pp. 84–91. DOI: 10.1109/SIS.2005.1501606.

[47] T Kalaiselvi et al. “A Review on Glowworm Swarm Optimization”. In: *International Journal of Information Technology (IJIT)* 3.2 (2015), pp. 49–56. URL: www.ijitjournal.org.

[48] Yuli Zhang et al. “Localization of multiple odor sources using modified glowworm swarm optimization with collective robots”. In: *Proceedings of the 30th Chinese Control Conference, CCC 2011* (2011), pp. 1899–1904.

[49] Thomas Wiedemann et al. “Model-based gas source localization strategy for a cooperative multi-robot system—A probabilistic approach and experimental validation incorporating physical knowledge and model uncertainties”. In: *Robotics and Autonomous Systems* 118 (2019), pp. 66–79. DOI: 10.1016/j.robot.2019.03.014. URL: https://doi.org/10.1016/j.robot.2019.03.014.

[50] Massimo Vergassola et al. “’Infotaxis’ as a strategy for searching without gradients”. In: *Nature* 445.7126 (2007), pp. 406–409. DOI: 10.1038/nature05464.

[51] Michael Hutchinson et al. “Entrotaxis as a strategy for autonomous search and source reconstruction in turbulent conditions”. In: *Information Fusion* 42.October 2017 (2018), pp. 179–189. DOI: 10.1016/j.inffus.2017.10.009. URL: https://doi.org/10.1016/j.inffus.2017.10.009.

[52] Christian Bilgera et al. “Application of convolutional long short-term memory neural networks to signals collected from a sensor network for autonomous gas source localization in outdoor environments”. In: *Sensors (Switzerland)* 18.12 (2018). DOI: 10.3390/s18124484.

[53] Hyunseung Kim et al. “Source localization for hazardous material release in an outdoor chemical plant via a combination of LSTM-RNN and CFD simulation”. In: *Computers and Chemical Engineering* 125 (2019), pp. 476–489. DOI: 10.1016/j.compchemeng.2019.03.012. URL: https://doi.org/10.1016/j.compchemeng.2019.03.012.

[54] William John Thrift et al. “Surface-enhanced raman scattering-based odor compass: Locating Multiple Chemical Sources and Pathogens”. In: *ACS Sensors* 4.9 (2019), pp. 2311–2319. DOI: 10.1021/acssensors.9b00809.

[55] Hangkai Hu et al. “Plume Tracing via Model-Free Reinforcement Learning Method”. In: *IEEE Transactions on Neural Networks and Learning Systems* 30.8 (2019), pp. 2515–2527. DOI: 10.1109/TNNLS.2018.2885374.

[56] Thomas Wiedemann et al. “Robotic information gathering with reinforcement learning assisted by domain knowledge: An application to gas source localization”. In: *IEEE Access* 9 (2021), pp. 13159–13172. DOI: 10.1109/ACCESS.2021.3052024.

[57] Yong Zhao et al. “A deep reinforcement learning based searching method for source localization”. In: *Information Sciences* 588 (2022), pp. 67–81. DOI: 10.1016/j.ins.2021.12.041. URL: https://doi.org/10.1016/j.ins.2021.12.041.

[58] Stephane Doncieux et al. “Evolutionary robotics: What, why, and where to”. In: *Frontiers Robotics AI* 2.MAR (2015), pp. 1–18. DOI: 10.3389/frobt.2015.00004.

[59] João Macedo et al. “A comparative study of bio-inspired odour source localisation strategies from the state-action perspective”. In: *Sensors (Switzerland)* 19.10 (2019), pp. 1–34. DOI: 10.3390/s19102231.

[60] B Gerkey et al. “The player/stage project: Tools for multi-robot and distributed sensor systems”. In: *Proc. of International Conference on Advanced Robotics (ICAR 2003)* Icar (2003), pp. 317–323.

[61] Olivier Michel. “WebotsTM: Professional Mobile Robot Simulation”. In: 1.1 (2004), pp. 39–42. URL: http://arxiv.org/abs/cs/0412052.

[62] Eric Rohmer et al. “V-REP: A versatile and scalable robot simulation framework”. In: *IEEE International Conference on Intelligent Robots and Systems* (2013), pp. 1321–1326. DOI: 10.1109/IROS.2013.6696520.

[63] Alexey Dosovitskiy et al. “CARLA: An Open Urban Driving Simulator”. In: CoRL (2017), pp. 1–16. URL: http://arxiv.org/abs/1711.03938.

[64] Epic Games. *Unreal Engine*. 2022. URL: https://www.unrealengine.com/.

[65] Shital Shah et al. “AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles”. In: *Springer Proceedings in Advanced Robotics* 5 (2018), pp. 621–635. DOI: 10.1007/978-3-319-67361-5{_}40.

[66] Nathan Koenig et al. “Design and use paradigms for Gazebo, an open-source multi-robot simulator”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* 3 (2004), pp. 2149–2154. DOI: 10.1109/iros.2004.1389727.

[67] Serena Ivaldi et al. “Tools for dynamics simulation of robots: a survey based on user feedback”. In: (2014), pp. 1–15. URL: http://arxiv.org/abs/1402.7050.

[68] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *IEEE International Conference on Robotics and Automation* (2009). DOI: 10.1109/IECON.2015.7392843.

[69] Jeongseok Lee et al. “DART: Dynamic Animation and Robotics Toolkit”. In: *The Journal of Open Source Software* 3.22 (2018), p. 500. DOI: 10.21105/joss.00500.

[70] Dongho Kang. *SimBenchmark*. 2021. URL: https://github.com/leggedrobotics/SimBenchmark.

[71] Marian Körber et al. “Comparing Popular Simulation Environments in the Scope of Robotics and Reinforcement Learning”. In: (2021). URL: http://arxiv.org/abs/2103.04616.

[72] Woong Gyu La et al. “DeepSim: A Reinforcement Learning Environment Build Toolkit for ROS and Gazebo”. In: (2022), pp. 1–10. URL: http://arxiv.org/abs/2205.08034.

[73] Yunlong Song et al. “Flightmare: A Flexible Quadrotor Simulator”. In: *Conference on Robot Learning* 4 (2020), pp. 1147–1157. URL: http://arxiv.org/abs/2009.00563.

[74] Arthur Juliani et al. “Unity: A General Platform for Intelligent Agents”. In: (2018), pp. 1–28. URL: http://arxiv.org/abs/1809.02627.

[75] Mario Coppola. “Automatic Design of Verifiable Robot Swarms”. PhD thesis. Delft University of Technology, 2021. DOI: 10.4233/uuid. URL: https://doi.org/10.4233/uuid:b6ad7ddd-c660-4aab-8277-65f7a22a4a52.

[76] NVIDIA Corporation. *Isaac Sim*. 2023. URL: https://developer.nvidia.com/isaac-sim.

[77] Jay A. Farrell et al. “Filament-based atmospheric dispersion model to achieve short time-scale structure of odor plumes”. In: *Environmental Fluid Mechanics* 2.1-2 (2002), pp. 143–169. DOI: 10.1023/A:1016283702837.

[78] D. R. Webster et al. “Laser-Induced Fluorescence Measurements of a Turbulent Plume”. In: *Journal of Engineering Mechanics* 129.10 (2003), pp. 1130–1137. DOI: 10.1061/(asce)0733-9399(2003)129:10(1130).

[79] N. S. Holmes et al. “A review of dispersion modelling and its application to the dispersion of particles: An overview of different dispersion models available”. In: *Atmospheric Environment* 40.30 (2006), pp. 5902–5928. DOI: 10.1016/j.atmosenv.2006.06.003.

[80] Daewon W. Byun et al. “Eulerian Dispersion Models”. In: *AIR QUALITY MODELING - Theories, Methodologies, Computational Techniques, and Available Databases and Software*. Vol. 1. EnvioComp Institute and Air & Waste Management Association, 2003. Chap. 10, pp. 213–291. DOI: 10.1007/978-1-4757-4465-1{_}6.

[81] J. Murlis et al. “Odor plumes and how insects use them”. In: *Annual review of entomology. Vol. 37* 86 (1992), pp. 505–532. DOI: 10.1146/annurev.ento.37.1.505.

[82] Eduardo Martin Moraud et al. “Effectiveness and robustness of robot infotaxis for searching in dilute conditions”. In: *Frontiers in Neurorobotics* 4.MAR (2010), pp. 1–8. DOI: 10.3389/fnbot.2010.00001.

[83] Qing Hao Meng et al. “Collective odor source estimation and search in time-variant airflow environments using mobile robots”. In: *Sensors* 11.11 (2011), pp. 10415–10443. DOI: 10.3390/s111110415.

[84] Gabriele Ferri et al. “SPIRAL: A novel biologically-inspired algorithm for gas/odor source localization in an indoor environment with no strong airflow”. In: *Robotics and Autonomous Systems* 57.4 (2009), pp. 393–402. DOI: 10.1016/j.robot.2008.07.004. URL: http://dx.doi.org/10.1016/j.robot.2008.07.004.

[85] Patrick P. Neumann et al. “Gas source localization with a micro-drone using bio-inspired and particle filter-based algorithms”. In: *Advanced Robotics* 27.9 (2013), pp. 725–738. DOI: 10.1080/01691864.2013.779052.

[86] Zeqi Li et al. “Assessment of different plume-tracing algorithms for indoor plumes”. In: *Building and Environment* 173.October 2019 (2020), p. 106746. DOI: 10.1016/j.buildenv.2020.106746. URL: https://doi.org/10.1016/j.buildenv.2020.106746.

[87] Alejandro Pequeno-Zurro et al. “A Chemosensory Navigation Model Inspired by the On/Off Neural Processing Mechanism in Cockroaches”. In: *IEEE Transactions on Medical Robotics and Bionics* 2.3 (2020), pp. 338–346. DOI: 10.1109/TMRB.2020.3007948.

A

Useful Links

This appendix contains several useful links for reference:

- GSL-Bench website: `https://sites.google.com/view/gslbench/`
- GSL-Bench video: `https://youtu.be/kZa48WXf_1w?si=GWGlhjDF6Q4d2KKO`
- AutoGDM+ Github: `https://github.com/tudelft/autoGDMplus`