2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

GSL-Bench: High Fidelity Gas Source Localization Benchmarking Tool

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 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 tool 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 Localization (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].

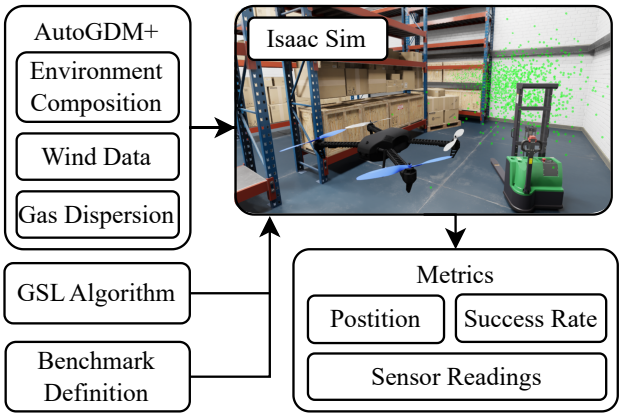

Fig. 1. System architecture of GSL-Bench. AutoGDM+ prepares the environments, wind and gas dispersion data. GSL-Bench performs the specified benchmark with the supplied GSL algorithm in Isaac Sim (gas filaments depicted as green dots), resulting in the metrics of interest.

Our contribution aims to alleviate these issues by making benchmarking more reliable, quicker, and simpler. In this paper, we present GSL-Bench, a benchmarking tool capable of benchmarking GSL algorithms, see Figure 1. Created with 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 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 tool 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.

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

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 as well. A Gaussian plume model depicts the time-averaged gas concentration as a function of location. It is straightforward 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 were 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 realistic visuals with a high-fidelity gas model.

A downside to Unity is the relatively 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 tool [26]. The structure of AvoidBench allows for swift implementation of one's obstacle avoidance algorithm and produces meaningful metrics based on 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 the 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.

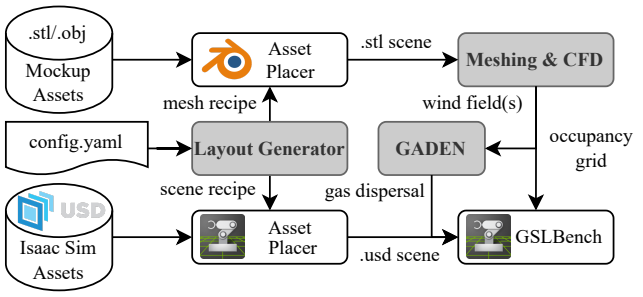

Fig. 2. Workflow of AutoGDM+. The respective asset placers create a scene from a generated layout. The .stl scene is used for meshing and CFD to generate wind fields and ultimately gas dispersion data. The .usd scene is used directly with GSL-Bench in Isaac Sim.

Environment generation starts with specifying the desired settings in a configuration file such as the type, size, and 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 Description (.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. These assets feature order of magnitudes less vertices but still capture the geometrical essence of the asset purely for CFD purposes. For now, the mockup assets are

created manually but this process can easily be automated with for example Blender.

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.

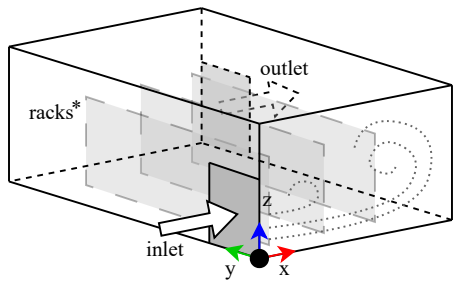

Fig. 3. CFD configuration of the warehouse environments. A velocity inlet and pressure outlet are situated near and far from the origin respectively. *Racks are only placed for the simple and complex warehouse types.

As the name implies, the empty warehouse features no interior. The simple warehouse includes rows of racks in the y-direction. The number of rows, their height, and length are 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 in 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 (VOCs) 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

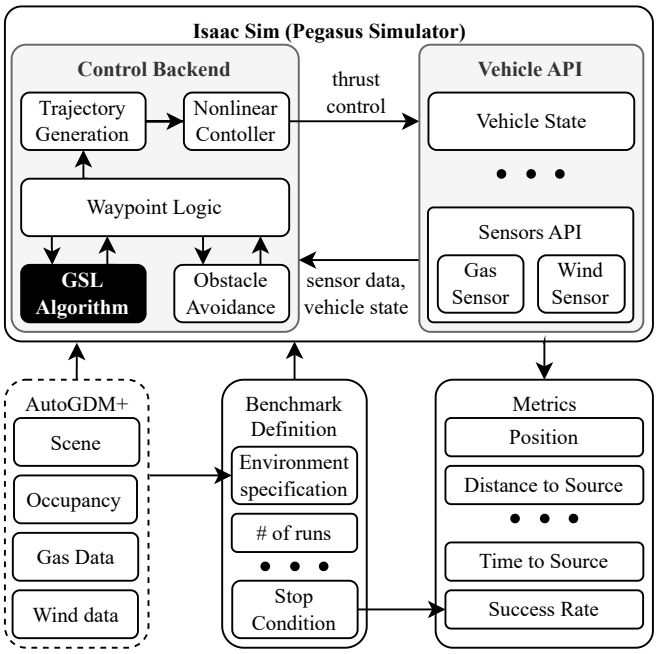

Fig. 4. System architecture of GSL-Bench. For Isaac Sim the Pegasus Simulator framework is used with the Python Control backend [25]. The generation of environments with AutoGDM+ is optional due to the availability of pre-generated environments. Only the GSL Algorithm (highlighted in black) is supplied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

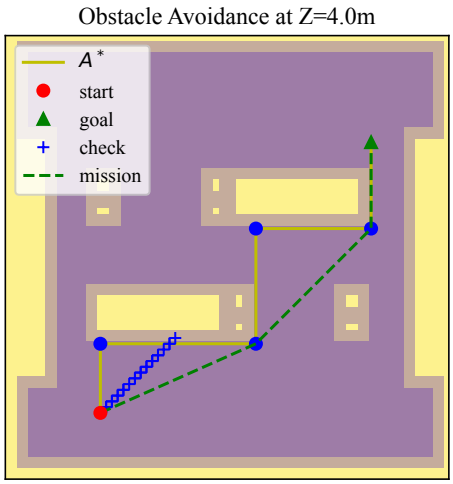

Fig. 5. Obstacle avoidance built into GSL-Bench. If the check runs into an obstacle, a mission is generated with an A^* algorithm.

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 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.

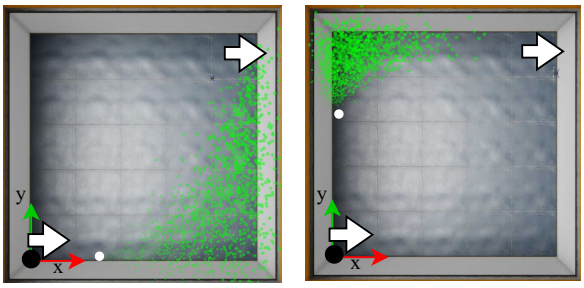

Fig. 6. Top-down view of environment 1 and 2 on the left and right respectively. The white arrows and dot indicate the in-/outlet, and source location. The green dots depict the gas filaments.

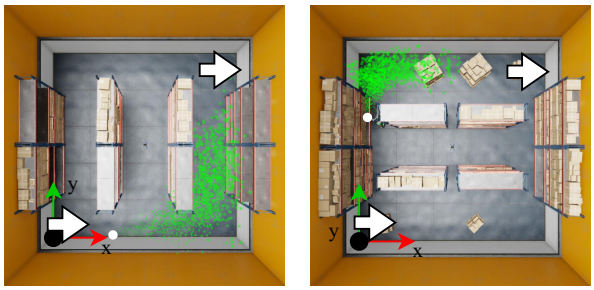

Fig. 7. Top-down view of environment 3 and 6 on the left and right respectively. The white arrows and dots indicate the in-/outlet, and source location. The green dots depict the gas filaments.

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 various experiments 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 is 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 number 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 modelled). 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 the 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 introduces 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.

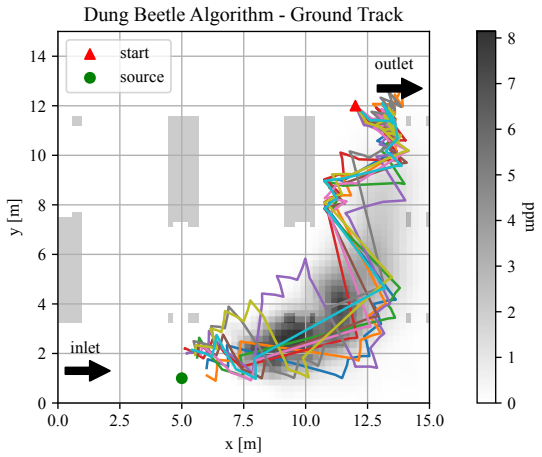

Fig. 8. Position traces of 10 Dung beetle runs in environment 3 from the upper right location. Obstacles are depicted as grey blocks. At first, the agents are stuck at the outlet but start zigzagging once some gas is detected.

1) Ground Tracks and Distance to Source: Ground track plots give an intuitive indication of an algorithm's behaviour. A good example is shown by Figure 8. The algorithm seems to struggle at first, but when the gas plume reaches closer to 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: The 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

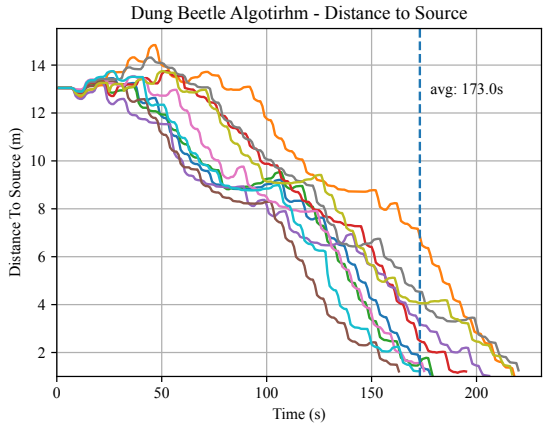

Fig. 9. Distance to source over time of 10 Dung beetle runs in environment 003 from the upper right location, highlighting the average time to the source.

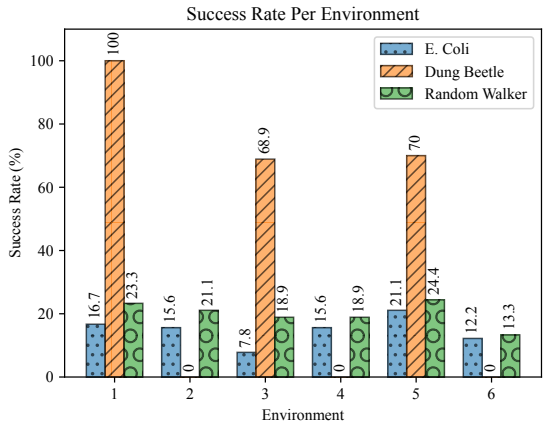

Fig. 10. Overall success rates per environment. The random walker outperforms the E. Coli algorithm in every environment, while the success of the dung beetle algorithm is heavily dependent on the source location.

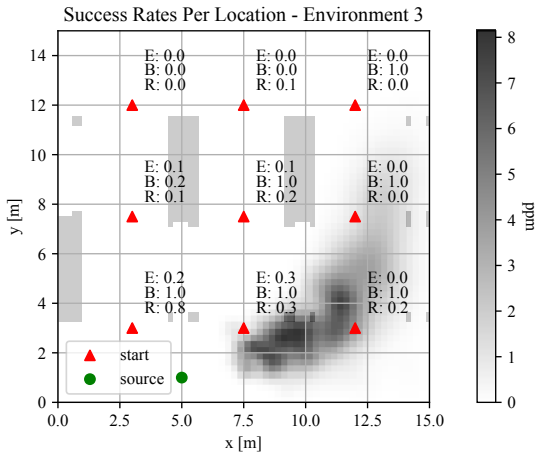

Fig. 11. Success rates for E. Coli (E), dung beetle (B), and random walker (R) for 10 runs per location in environment 3. Obstacles are depicted as grey blocks

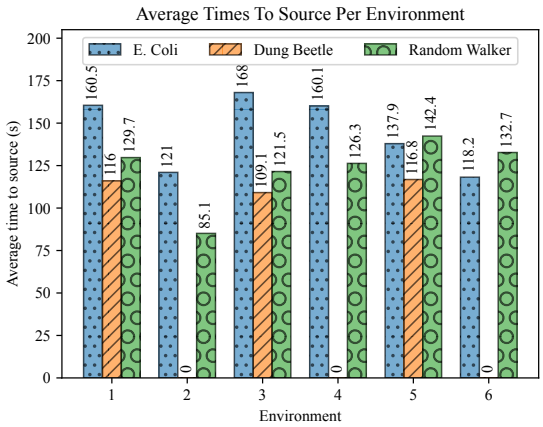

Fig. 12. Average time to source per environment. 0 indicates the absence of a measurement because no runs located the source.

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 the success rate to provide a meaningful indication. The performance of the dung beetle algorithm is consistent when it can 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 tool. 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.

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.