

Copyright (C) 2025 Aritra K. Mukhopadhyay.

README.md

- Project: Automated decision-making by chemical echolocation in active droplets
- Authors: Aritra K. Mukhopadhyay, Ran Niu, Linhui Fu, Kai Feng, Christopher Fujta, Qiang Zhao, Jinping Qu, and Benno Liebchen.
- Description: Motile microorganisms, like bacteria and algae, unify abilities like self-propulsion, autonomous navigation, and decision-making on the micron scale. While recent breakthroughs have led to the creation of synthetic microswimmers and nanoagents that can also self-propel, they still lack the functionality and sophistication of their biological counterparts. This study pioneers a mechanism enabling synthetic agents to autonomously navigate and make decisions, allowing them to solve mazes and transport cargo through complex environments without requiring external cues or guidance. The mechanism exploits chemo-hydrodynamic signals, produced by agents like active droplets or colloids, to remotely sense and respond to their environment - similar to echolocation. Our research paves the way for endowing autonomous, motile synthetic agents with functionalities that have been so far exclusive to biological organisms.
- Contact: aritra.mukhopadhyay@pkm.tu-darmstadt.de, benno.liebchen@pkm.tu-darmstadt.de

What this project does

- Runs time-dependent simulations of a scalar chemical field coupled to particle motion inside maze-shaped domains.
- Supports moving and static point sources, optional advection, chemotactic response, particle-particle interactions (Lennard-Jones style), wall avoidance, and self-propulsion.
- Writes per-timestep concentration (`data/conc_*.txt`) and particle (`data/part_*.txt`) outputs and can render a trajectory video (`data/particle_trajectory.mp4`).

File descriptions

- `maze_cluster_script.py` — example driver that sets simulation parameters, generates/loads a maze, runs the solver loop and saves outputs to `data/`.
- `list_of_functions.py` — numerical routines and I/O helpers (solvers, force models, write functions).
- `maze_functions.py` — maze generators and loaders (random, box, custom, load from TSV).
- `video_maker.py` — reads `data/` files and renders an MP4 animation of concentration + particle trajectories.
- `master_run.sh` — simple helper to run the cluster script followed by the video maker.

How to run the simulations

Prerequisites

- Python 3.8+ (code was tested on Python 3.10+). Ensure you have a working Python environment with the scientific stack.

- Recommended packages: `numpy`, `matplotlib`, `seaborn` (for the video script), and `ffmpeg` (for saving MP4s).

Install minimal dependencies (example using pip):

```
python -m venv .venv
source .venv/bin/activate
pip install --upgrade pip
pip install numpy matplotlib seaborn
# ffmpeg should be available on PATH for `video_maker.py` to save MP4s
```

Run an example simulation

```
# From the repository root
./master_run.sh

# Or run directly
python maze_cluster_script.py
python video_maker.py
```

By default the example driver writes outputs into `data/` (see `maze_cluster_script.py` for parameter settings). Outputs include:

- `data/param.txt` — recorded simulation parameters
- `data/grid.txt` — grid description
- `data/conc_*.txt` — concentration snapshots
- `data/part_*.txt` — particle snapshots
- `data/particle_trajectory.mp4` — rendered video (created by `video_maker.py`)

Configuration and parameters

- `maze_cluster_script.py` contains the main parameter block (diffusion coefficients, time step `dt`, `n_steps`, `dx`, particle counts, source strengths, etc.). Edit that file to experiment with different regimes.
- `maze_functions.py` provides multiple maze sources: random generator, `box_maze`, `custom_maze`, and `maze_from_file(file.tsv)` to load maze TSVs in `different_mazes/`.

Example: change grid size and time steps in `maze_cluster_script.py`:

```
# change domain size and resolution
dx = 1.0
Lx = 200.0
Ly = 200.0
n_xbins = int(Lx/dx)
n_ybins = int(Ly/dx)
n_steps = 4000
```

Output format

- Concentration files (`data/conc_<t>.txt`) use a small header then a flat numeric matrix that `numpy.loadtxt(..., skiprows=3)` can read. See `list_of_functions.write_concentration` for the writer format.
- Particle files (`data/part_<t>.txt`) include a header and columns: `particle_id x y theta vx vy omega f_spx f_spy f_chemx f_chemy f_intx f_inty f_wallx f_wally`.