VPS: Brownian Motion Particle Video Simulator

Clear Documentation

Yagna

September 29, 2025

Abstract

This document explains the design and usage of a simple Brownian-motion video simulator implemented in Python. The `VPS` class simulates microscopy-like image sequences of diffusing particles, provides an optional Matplotlib animation, and includes basic analysis tools (e.g. mean squared displacement). We cover the physical model, implementation details, parameters and defaults, and examples. The document is intended for inclusion in Overleaf; copy this file as `main.tex` and compile with LaTeX.

1 Overview

The simulator creates a grayscale image frames that mimic particles under-going two dimensional Brownian motion inside a rectangular field of view. Each frame is generated by placing Gaussian intensity profiles at particle coordinates (optionally blurred and with background noise), then writing the stack to make a video. A companion analysis routine compares the simulated mean-squared displacement (MSD) with the theoretical values.

Key features

- Physical parameters set in SI units (pixel size, temperature, viscosity, particle radius, frame rate).
- Diffusion coefficient via Stokes–Einstein; step variance linked to physics and frame time.
- Defined boundaries with reflected edges (no loss of particles)
- Multiple initial spatial distributions (`random`, `center`, `grid`).
- OpenCV video export and Matplotlib-based live animation of trajectories.
- Built-in MSD computation.

2 Physical Model

2.1 Brownian Motion and Stokes–Einstein

For a spherical particle of radius r in a Newtonian fluid with dynamic viscosity η at absolute temperature T, the diffusion coefficient D is

$$D = \frac{k_B T}{6\pi\eta r}, \tag{1}$$

where k_B is the Boltzmann constant, r is the radius of the particle, T is the temperature of the liquid, and η is the viscosity.

Over a frame interval Δt, a 2-dimensional Brownian step $(\Delta x, \Delta y)$ is modeled as independent Gaussian increments.

$$\Delta x, \Delta y \sim \mathcal{N}\big(0,\, \sigma^2\big)\,, \qquad \sigma = \sqrt{2D\Delta t}. \tag{2}$$

Steps are computed in meters and converted to pixels using the calibrated pixel size p (m/pixel).

2.2 Mean-Squared Displacement (MSD)

The ensemble MSD for 2D diffusion grows linearly with time:

$$\mathrm{MSD}(t) = \langle \Delta r^2 \rangle = 4Dt. \tag{3}$$

The analysis function estimates MSD from simulated trajectories and overlays the theoretical line for validation.

3 Simulator:

3.1 Dependencies

- `numpy`, `scipy` (`scipy.ndimage`),
- `matplotlib.pyplot`,
- `matplotlib.patches` (`Circle`),
- `matplotlib.animation` (`FuncAnimation, PillowWriter`)
- `opencv-python` (as `cv2`).

3.2 Core Concepts

State Image geometry, physical constants, rendering options, and derived quantities (diffusion coefficient, step size in meters and pixels, particle radius in pixels).

Update At each frame, independent Gaussian steps are drawn for every particle; positions are updated and reflected at boundaries.

Render Particles are rendered as additive Gaussian blobs, optional Gaussian blur models optics, and background noise mimics sensor noise.

4 Quick Start

4.1 Minimal Example

```python
from vps import VPS  # if saved to vps.py

sim = VPS(
    image_size=(512, 512),
    pixel_size=100e-9,
    temperature=298,
    viscosity=1e-3,
    particle_radius=500e-9,
    fps=30,
    particle_brightness=0.7,
    background_noise=0.02,
    blur_sigma=0.5
)

frames, positions = sim.simulate_video(
    n_particles=20,
    n_frames=300,
    output_filename="brownian_particles.mp4",
    particle_distribution="random"
)

ani = sim.create_matplotlib_animation(frames, positions, show_trails=True)
sim.analyze_simulation(positions)
```

4.2 Environment Setup

```
pip install numpy scipy matplotlib opencv-python
```

5 Parameter Reference

All inputs are SI-based where applicable. Table 1 summarizes constructor arguments.

Name	Type	Default	Description
`image_size`	tuple(int,int)	(512,512)	Frame height and width in pixels.
`pixel_size`	float	1×10^{-7}	Physical size per pixel (m/pixel).
`temperature`	float	298	Absolute temperature T in K.
`viscosity`	float	0.001	Dynamic viscosity η in Pa s.
`particle_radius`	float	1×10^{-6}	Particle radius r in m.
`fps`	float	30	Frames per second; sets $\Delta t = 1/\text{fps}$.
`particle_brightness`	float	0.8	Peak additive intensity of each particle (0–1).
`background_noise`	float	0.05	Std. dev. of additive Gaussian background noise.
`blur_sigma`	float	1.0	Image-space Gaussian blur (pixels).

Table 1: Constructor parameters for `VPS`. Internally, $k_B = 1.380\,648\,52 \times 10^{-23}\,\text{J/K}$.

6 API Documentation

6.1 `VPS.initialize_particles(n_particles, distribution)`

Returns initial positions as an $(N, 2)$ array in pixel coordinates.

`For distribution` uses one of `"random"`, `"center"`, `"grid"`.

6.2 `VPS.update_particle_positions(positions)`

Adds positions by one frame using Gaussian steps with std. $\sigma_{\text{pix}} = \sigma_{\text{m}}/p$. Applies reflection at all boundaries.

6.3 `VPS.render_frame(positions)`

Creates one grayscale image frame by drawing soft round spots for each particle, then optionally adds background noise and a blur effect.

6.4 `VPS.simulate_video(n_particles, n_frames, output_filename, show_trails, particle_distribution)`

Runs a complete simulation and returns:

- `frames`: a stack of simulated images with shape (T, H, W), where
  - T = number of frames (time steps),
  - H = image height in pixels,
  - W = image width in pixels.

  Pixel values are between 0 and 1 (grayscale).
- `all_positions`: particle coordinates with shape $(T, N, 2)$, where
  - T = number of frames,
  - N = number of particles,
  - $2 = (x, y)$ coordinates in pixels.

If `output_filename` is given, the simulation is also saved as a video file (MP4 using `mp4v` or AVI using `XVID`) via OpenCV.

6.5 `VPS.save_video(frames, filename)`

Writes video to disk at the simulator frame rate using OpenCV; grayscale frames are converted to 8-bit BGR for encoding.

6.6 `VPS.create_matplotlib_animation(frames, all_positions, show_trails)`

Creates a two-panel Matplotlib animation: left shows the evolving image; right shows trajectories (up to 10 for clarity). Returns a `FuncAnimation` object suitable for interactive display or saving.

6.7 `VPS.analyze_simulation(all_positions)`

Computes the MSD versus time and compares to $4Dt$. Also plots sample trajectories, the step-size distribution, and the final rendered frame with a 5 µm scale bar. Prints summary statistics including theoretical step size per frame and observed mean step size.

7 Units and Conventions

- Positions are stored in *pixels*; conversions use `pixel_size` to obtain meters or microns for plotting.
- The Gaussian "particle radius" in pixels is r/p; the visual blob width is set by $(r/p)/2$ inside `render_frame`.
- Blur σ is in *pixels*; background noise is unitless intensity added before clipping to $[0, 1]$.

8 Validation and Interpretation

1. Ensure the measured MSD slope is close to $4D$; large deviations suggest unit mismatch or excessive clipping/noise.
2. Check that typical per-frame step sizes $\sigma_{\mathrm{m}} = \sqrt{2D\Delta t}$ are small relative to the field of view but not vanishingly small.
3. If particles appear to "stick" at boundaries, remember reflection is used; periodic boundaries can be added as an extension.

9 Troubleshooting

Video is too dark/bright Adjust `particle_brightness`, `background_noise`, and `blur_sigma`.

Particles look too big/small Tweak `particle_radius` or the rendering width inside `render_frame`.

Computation is slow Reduce frame size, number of particles, frames, or disable animation. Vectorizing the rendering step or using FFT-based splats can further accelerate.

OpenCV missing Install `opencv-python`. On headless servers, saving animations via Matplotlib may require `ffmpeg`.

10 Extending the Simulator

- **Heterogeneous particles:** per-particle radii or diffusion coefficients.
- **Forces/flows:** add drift, confinement, or interactions.
- **Noise model:** shot noise, camera read noise, spatially varying background.
- **Optics:** replace Gaussian PSF with measured or Airy PSF.
- **Boundaries:** periodic or absorbing boundary conditions.

11 Reproducibility

To reproduce the example, set a random seed before constructing the simulator:

```
import numpy as np
np.random.seed(42)
```

Acknowledgements

If you use this simulator in a report or paper, please cite this documentation and the libraries.