Plume Model 2.0

in-house explanation and how to

Nathaniel Saul

1 Intro

The goal of this plume simulation is to generate a model that will suffice in testing and development of plume tracking algorithms. Standard plume modelling methods are potentially unsuitable for this algorithm development (Jones 1982, Elkinton 1983). A satisfactory plume simulation has to resemble a plume both instantaneously and over a time-averaged view. To accomplish this, the plume simulation presented here uses a random walk model (Kinzelbach 1990, Kitandis 1994). In the limit, the random walk simulates a Gaussian advection-diffusion plume, but when tuned down, with the number of molecules being far from the limit, shows high intermittency and erraticness seen in (Jones 1982 and Elkinton 1983).

1.1 Requirements:

1. Instantaneously, the plume will have a high intermittency. For the majority of time (upwards of 80%) there is a material concentration of zero at any given point. (Jones1983).

2. The long-term, time-averaged plume, will resemble a plume model such as the model used in the original simulation.

3. The simulation will be able to retrieve concentration, gradient, divergence of gradient, and flow vector at all locations (x, y) and all time t.

2 Governing Equations

2.1 Random Walk

The random walk equation (1) has been derived by Kinzelbach (1990) from the transport equation of an ideal tracer. In the limit, as both $\Delta t \to 0$ and number of molecules $\to \infty$, the concentration of equation (1) will be equivalent to the advection-diffusion equations solved in the original environment model.

$$x_{t+\Delta t} = x_t + u\Delta t + Z(2\alpha_L u\Delta t)^{1/2} \tag{1}$$

Here u represents the fluid velocity vector, Z is a normally distributed random variable centered at 0 and with a variance of 1, α_L is the longitudinal dispersivity, or the diffusion coefficient and t is time.

2.2 Concentration

The concentration at any point x is then the total number of molecules m within a small radius $r$of the point x, divided by a normalizing factor based on the area of observation and the water and molecule properties.

$$c(x,t) = card(\{m : ||m - x|| < r\})$$

3 LCM Interface

The environment thread consists of 2 main functionalities, environmental update and data retrieval. The lcm channels to cue these are respectively named *envUpdate* and *envRetrieve*.

envUpdate will push the plume forward 1 time-step, dt, and return a message on channel *envUpdateConfirm* when it is finished. It does not matter what is in the message that is sent in either direction, they are only used as cues. *envRetrieve* takes a point in Cartesian coordinates and returns the estimated concentration, gradient of concentration, the normal vector to the gradient, divergence of the gradient, and the fluid flow velocity.

Input *envRetrieve*	Hidden	Output	*dataReturn*
(x,y)	C(x,y)	Concentration	U0
	C(x+dx, y)	Gradient	DU
	C(x-dx, y)	Divergence	D2U0
	C(x, y+dy)	Normal to gradient	DU_p
	C(x, y-dy)	Fluid flow vector	V0

The current implementations of calculations for the gradient and the divergence require five points of concentration to be surveyed. Currently, we use points up, down, left, and right of the robot.

4 Operation

First, download the code from the github repository:

`https://github.com/sauln/plumeSimulation`

4.1 Simulation with lcm

To run the plume in the loop with the robot simulation, you must run 3 files:

1. *plumeSim/plumeSimWlcm.py*,
2. *roboSim/controlThread.py* and

3. *roboSim/simulationThread.py.*

plumeSim/plumeSimWlcm.py must be run within *ipython –pylab* for the visualizations to work properly. The simulation is currently setup to use the single integrator robot model as this was the last stable version I have on my machine. Plugging in the new model should be as simple as changing the lcm channel names.

4.2 Helpful functions

To run the plume simulation by itself, you can run *plumeSim/plumeSim.py.* This file is currently setup to run the basic plume model, but there are a number of functions in *plumeSime/plumeExperiment.py* that explore different random walk equations and calculate various statistics and interesting plots about the plumes. Most important are the functions that generates statistics about the intermittency of the plume and the time-averaged probability distribution function at various points down the plume.

4.3 Fluid flows

There are 2 different fluid flows that you can use. There is also a lot of room to add new fluid flows. When the simulation initiates the plume, it defines which flow to use for the calculations. The two that are built in now are named 'simple' and 'mit'. The 'mit' is the same fluid flow that was used in the original plume simulation. The 'simple' flow is a constant downstream flow.

5 Initial Observations

In my experiments, it is very clear that something is wrong. I found a few bugs with how the gradient and divergence of gradient were calculated, but now there are a few test functions that show the calculations are correct now.

The main observation is it seems that when the robot looses touch with the plume (all sensors read zero), the robot continues in the previous direction it was moving. Intuitively, I would expect it to turn around at that point in an attempt to re-establish contact.

6 Todo

1. Calibrate the plume. Currently the calibration is only estimated. This includes the concentration normalizing factor, how many molecules to emit, and the relative time frame.

2. Move the gradient and divergence calculations to the *estimator* side of the simulation.