UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DEL GRAU EN ENGINYERIA FISICA

Simulating Underwater Molecular Propagation Signal

Author:
Xabier GUTIÉRREZ de la CAL

Supervisor:
Dr. Ilker S. DEMIRKOL

June 25, 2018

Contents

UPC, UNIVERSITAT POLITÈCNICA DE CATALUNYA

Abstract

ETSETB, Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona
Department or School Name

Undergraduate in Engineering Physics

Simulating Underwater Molecular Propagation Signal

by Xabier GUTIÉRREZ de la CAL

Molecular communication via difussion (MCvD) is a molecular communication (MC) system alternative to traditional communication, in environments where the latter are inefficient to propagate a signal.

For macro-scale topologies, the huge amount of molecules to have a detectable signal force us to think about a solver which does not simulate each molecule individually; in the following work a voxel-based tracking solver is proposed and evaluated, compared with a particle-based approach, and optimized to generate a signal as close as possible to this, but being computationally less expensive.

The relation between parameters of the MCvD environment and optimal voxel length for the simulator is analyzed.

Acknowledgements

First, I want to thank deeply the guidance during the development of this project of Dr. Ilker S. Demirkol, who proposed this subject and help me introduce a little in the world of Molecular Communications. Besides the supervision in the project itself, he has helped in many formal aspects for the realization of this kind of investigation, that I have not done before, and the composition of the thesis.

I would like also to acknowledge the immense contributions from Dr. H. Birkam Yilmaz, who showed great passion for the subject and the development and implementation of the simulators, helped during the many doubts and sometimes errors that arose during the realization of the project and without whom it would not have been possible.

To both of them I thank the opportunity to be part of such a project, the educational it has been and also being able to see my contributions accepted and part of future work.

I want to thank my family for the support during this last years of study: I want to thank my friends, who after this years away from home remain and to whom I am immensely grateful, and to the friends and people I discovered living abroad, and have been of great support.

Chapter 1

Introduction

1.1 The need for Molecular Communication (MC)

Communication is one of the most important aspects of modern human society, and great technological effort is put in pushing its actual limits as further as possible. This is done by improving current communication systems, making them more efficient and their reach to be expanded, as well as bringing these technologies to face new challenges. Nevertheless, the traditional methods of communication (electromagnetic, acoustic and optical communication systems) present some inefficiencies when facing certain problems, the two main of which are:

- **Small dimensions** : Recent advances in nano and biotechnology opens the door to devices adapted to those scales, and communication systems with them [5]. Nevertheless, it has been demonstrated that for channels small enough, or with many obstacles in it, EM wave communication are inefficient to propagate [6].
- **Specific environments** : In certain environments, such as networks of tunnels, pipelines, or salty water environments, traditional communication signals (wave, acoustic, optical) fail to propagate efficiently, due to the high losses. At sea, for example, the high salinity produces conductivity and therefore attenuation of the sent signal, and optical waves suffer from scattering [9].

This makes obvious that, for microscale and nanoscale communication, new devices and methods are needed. One of the options is **Molecular Communication (MC)**; which consists on using chemical signals to carry the information. This type of communication already happens in nature: A variety of animals use different kinds of chemicals to communicate, both in macro and micro-scales, and communications inter and intra-cellular [3], pheromones are used by members of the same species at long range [1]; this examples can be looked to as a possible inspiration.

Besides the mentioned advantages in small dimension channels or specific environments, MC is also a biocompatible method of communication, as it already is present in nature, requires very little energy and has very low heat dispersion, being therefore a strong alternative for certain applications. The study of this field is, still, very novel, as it has only recently started [2].

1.2 Overview of MC

MC, as any other communication method, transfers information across time and space, with the three same major components: a signal is generated at the **transmitter** and travels across the **channel** to the **receiver**, where it is decoded.

The signal is made up by **information particles**, typically of few nanometers of size, that can be of very different structure and size, biological or synthetic compounds, and even of some complexity (proteins or liposomes, for example). The kind of particle will affect the communication, as they affect the diffusion coefficient and may degrade with time [7].

If in EM communication the channel is a wire or the free space, in MC these channels, through which the information particles propagate, is typically an aqueous or gaseous environment.

Micro and **macro-scale** MC can be differentiated depending on the distance employed, being from nm to cm and from cm to m, respectively. Both present different communication properties and methods of propagation, since physical properties vary with the scales.

The mechanisms through which the signal travels can also vary between different MC systems. They can be categorized in **diffusion based propagation**, **flow assisted propagation**, **active transport using molecular motors and cytoskeletal filaments**, **bacterial assisted propagation**, and **kinesin molecular motors moving over immobilized microtubule (MT) tracks** [5].

As to decode the received signal, micro and macro-scale communications can be differentiated, since in macro scale communication single particles may be too difficult to detect, and the concentration is the focus in order to have a detectable signal. Any measurable parameter can serve to decode: Presence or abscence of molecules, levels of concentration (threshold), type of particles arriving at the receiver, time of arrival,... Choosing any of these will depend on the characteristics of the environment and the purpose of communication.

The focus of this project will lay in diffusion based MC, or **Molecular Communication via Diffusion (MCvD)**.

1.2.1 Molecular Communication via Diffusion (MCvD)

Also referred as Brownian motion, it is the random motion of a particle due to collisions with other particles, causing a group of particles to propagate to its surroundings, and diffuse to it. When the number of particles is high enough, as in actual MCvD systems, this diffusion allows a detectable signal to propagate.

Diffusion based MC is the most simple, energy efficient and lacks of external force. It is important to notice that the particles propagate uncontrolled, or at least only partially controlled, in the medium.

A typical MCvD environment in a free space is depicted in Fig. 1.1.

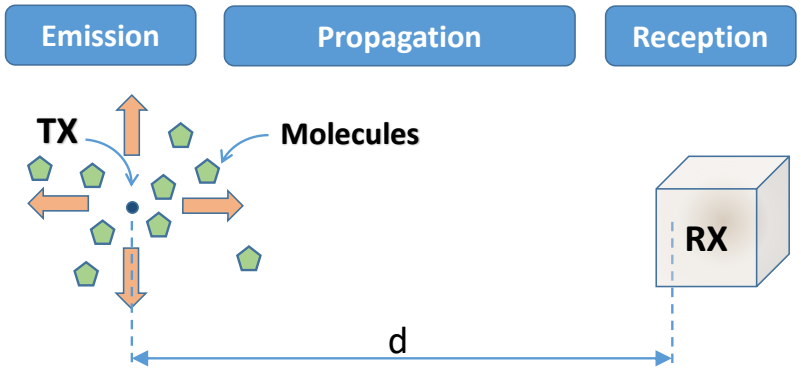


Figure 1.1: MCvD environment: transmitter (TX), receiver (RX) and molecules.

1.3 Current state of the art

MC has only recently been started to be investigated, as the need for a smaller size communication devices and the unsuitability of traditional electronic or electromagnetic devices, miniaturized, for such tasks became apparent.

Some of the proposed applications to its use is to build nanonetworks that connect nano-machines, set of molecules able to perform simple tasks [2]. Their interconnection enables a group of nano-machines to perform tasks more complex than those a single one is able to.

Underwater wireless communications are actually needed for tasks such as remote control in off-shore oil industry, pollution monitoring in environmental systems and collection of scientific data [9].

Analytical solutions for simple MCvD might be found, for example, for perfect absorbent spheric receiver [10], or reversible absorption receiver [4], but for more complex topologies the use of simulations are needed.

Due to the complexity of many of the aspects of a big enough MC system, Machine Learning techniques have been used in order to obtain useful models [8].

1.4 Project objectives and overview

This project has as a main objective to develop an alternative simulation solver for MCvD, computationally less expensive than particle-based simulations, which can be, for big enough simulations, too expensive. The alternative that is proposed is a **voxel-based simulator**.

2D and 3D versions of it are developed, and evaluated its accuracy to changes in physical parameters of the environment and simulation parameters, which are the **diffusion coefficient, D**, the **distance to the receiver, d** and the **time step,** δt. Specifically, the optimal length, λ, for the voxels is looked for. Besides this, flow is added to the simulations, to analyze how it affects the optimal voxel length.

The content of this thesis has been structured in the following way:

- **Chapter 2: Background.** The main physical aspects related to MCvD are explained: Fick's law of diffusion and Brownian motion.
- **Chapter 3: Development of the work.** The methods of the thesis, the basics of how the simulators work, the two simulation solvers used, particle-based and voxel-based, are explained, and how these have been used to obtain the necessary data to analyze them. Also, which tools are used to analyze this data are exposed.
- **Chapter 4: Results and discussion.** The results are presented accompanied by a discussion of its significance within the study.
- **Chapter 5: Conclusions.** Some final remarks about the main results of the project, and some possible future work based on them.

Chapter 2

Background

2.1 Physical fundamentals

MCvD deals with the random movement of small particles in a fluid medium; we will take two different approaches to this: Microscopic, taking into account the particles individually, which follow **brownian** dynamics , and macroscopic, modeling the whole system and its **diffusive** behavior. The microscopic movements result on the macroscopic ones.

2.1.1 Fick's law of diffusion

The macroscopic approach focuses on the **concentration** of particles, their distribution in space and time, and studies its change. Another important parameter in the macroscopic theory of diffusion is the **flux**, which is defined as the amount of unit particles that crosses a unit surface per unit time.

Fick's laws of diffusion model and describe relations between this magnitudes in a diffusive system.

It postulates that the flux moves from high to low concentration regions, in steady state we have that it is linearly proportional to the concentration gradient. This is **Fick's first law**:

$$J = -D\nabla C(\vec{r}, t) \tag{2.1}$$

where J is the flux, $C(\vec{r}, t)$ the concentration at point $\vec{r}$ and time t and D the diffusion constant. D is a positive constant, and therefore the minus sign represents that the flux, and particles with it, goes in the direction of decreasing concentration.This equation can be derived taking a particle performing a **random walk**, and $D = \frac{\Delta x^2}{2\Delta t}$, which is the definition of the constant.

As to the change with time, this flux forces a change in the spatial distribution of the concentration, since particles, as seen, move with it, causing a change in the flux itself: We use the **mass conservation equation**, which states:

$$\frac{\partial C}{\partial t} + \frac{\partial J}{\partial x} = 0 \tag{2.2}$$

From which, using (2.1) and assuming D is a constant (and therefore $\frac{\partial D}{\partial x} = 0$), we can get **Fick's second law**:

$$\frac{\partial C(\vec{r},t)}{\partial t} = D\Delta C(\vec{r},t) \tag{2.3}$$

where the Laplacian operator, $\Delta(=\nabla^2)$, generalizes the second spatial derivative to any dimension.

The first Fick's law can be seen as a steady state version of the second one. Fick's laws can be understood as stating that particles will tend to be more homogeneously distributed in space: If molecules start spatially concentrated one near the other, as time goes on they will move separately from one another, and fill the space around them, which is the diffusion itself.

This can be seen solving (2.3); with an initial concentration of N particles in the origin ($C(\vec{0},0)$ = N) it gives a solution of the form:

$$C(\vec{r},t) = \frac{N}{\sqrt{4\pi Dt}} e^{-\frac{\vec{r}^2}{4Dt}} \tag{2.4}$$

Which is a normal distribution of mean $\mu = 0$ and variance $\sigma = \sqrt{2Dt}$. This solution shows, as time increases, how the initial peak of concentration in $\vec{x} = \vec{0}$ tends to be more and more distributed in space.

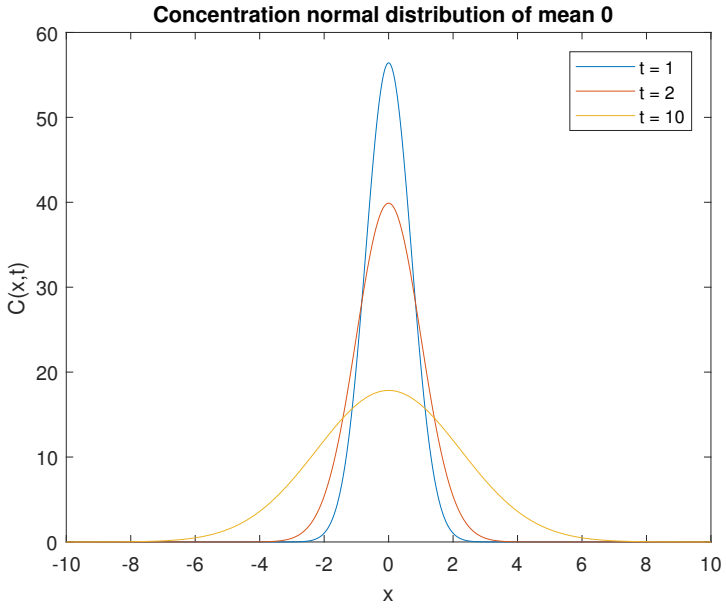


Figure 2.1: Example of normal distribution

This can be seen in Fig. 2.1, as for larger times the initial distribution diffuses in space from the initial peak concentrated in $x = 0$ (values and units of the figure have no special meaning since it is only an example).

2.1.2 Brownian motion

From a diffusive microscopic point of view, molecules move individually but still moving away from each other in a certain concentration. Their movements are generated by the interaction between one another, which, for a

concentration high enough, meaning many interactions between individual particles, can be taken as random.

Focusing on the particles, a discretized version of the normal distribution solution (2.4) can be taken as the probability of displacement of the random motion, meaning that for a unit of time, the particle has moved a random distance in a random direction, and each of the distances in the axis chosen, therefore, the directions, are taken from a normal distribution, which depends on the unit of time.

Chapter 3

Development of the work

3.1 Simulation

MCvD deals with two devices, a transmitter and a receiver, in a fluid environment, which is assumed to be unconfined: The information is carried by the molecules, from the transmitter to the receptor. There are many possibilities for the topology, depending on the transmitter, receiver and the environment. Systems may be too complex to obtain an analytical solution, and simulations become a necessity, especially as the system grows in size.

Time is discretized, with δt being the unit of time: Every time step, particles move in the environment.

The focus of the simulations will be the concentration arriving at the receptor, $N_{R_x}(t)$, which will constitute the signal and will be dependent on time: For a given window of time , between a certain time t_0 and $t_1 = t_0 + \delta t$, $N_{R_x}(t_1)$ will be the number of particles that have arrived to the receptor, starting in t_0, in a lapse of time of δt. This, for all time steps of the simulation duration, will be the received signal.

With $C(V_{R_x}, t_i)$ being the concentration at a certain volume, or area, V_{R_x}, occupied by the receiver in its positions, and at a certain time, t_i, the signal in the receiver at this time will be:

$$N_{R_x}(t_{i+1}) = C(V_{Rx}, t_{i+1}) - C(V_{R_x}, t_i) \tag{3.1}$$

We expect the signal to increase its value, have a certain peak coinciding with a peak in the concentration at that point, and then decrease as time goes on and the molecules diffuse in the medium. The receiver is assumed to be perfectly absorbing, therefore particles that arrive to it are removed from the environment.

Since $C(V_{R_x}, t_i)$ depends directly on V_{R_x}, $N(t_i)$ will depend on the topology of the receptor.

The emitter's topology may also be of certain importance, but for a distance enough between it and the receptor this effect will not be significant. For the shake of the simulations, it can be assumed to be a point in space, where all particles are concentrated before the initiation. As communication starts, the particles will move in every direction in space, following the laws of diffusion.

Two simulation approaches can be made:

- **Particle-based:** Each particle's trajectory is simulated individually. The computational time complexity of this solver depends on the number of molecules.
- **Voxel-based:** Space is divided into voxels (squares or boxes in 2D or 3D, respectively) with particles homogeneously distributed in each of them, and only the number of particles in each voxel is simulated, making its computational time complexity independent of the number of particles, and dependent on the size of the environment simulated.

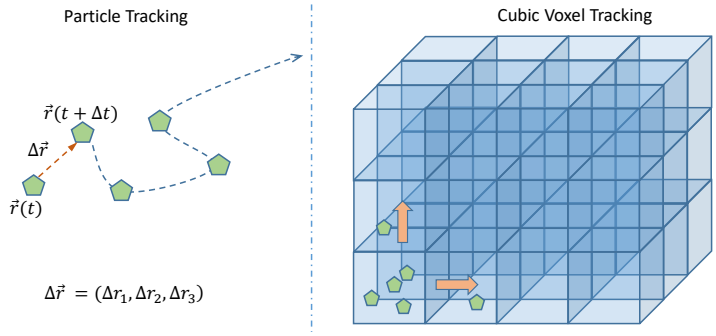


Figure 3.1: Both simulation approaches for a 3D MCvD system

For the sake of simplicity, interactions between different particles, such as collisions, are not directly taken into account in any of both the solvers.

For systems big enough, the number of molecules has to increase to deal with the loses of the medium in order to have a detectable and analyzable signal, and, considering these two approaches, voxel-based is to consider as an alternative due to its runtime independence of the number of particles.

3.1.1 Particle-based simulators

Simulating the brownian motion of the particles can be seen as a discretized version of diffusion. As seen before in (2.4), brownian particles, starting from a point, diffuse as a Gaussian normal function. For the simulations, the displacement of each particle in the fluid medium, in a certain step of time, of duration δt, which is random displacement, is taken as Gaussian:

$$\Delta x \sim N(0, \sigma^2) \tag{3.2}$$

where Δx represents the increase in the position, x, in a time δt. This is a normal distribution of mean 0 and variance $\sigma = \sqrt{2D\delta t}$.

For a multidimensional simulation (2D and 3D), two or three independent random normal displacements are simulated, which constitutes the total displacement of the particle in a time step, $\Delta\vec{r}$. In 3D:

$$\begin{aligned} \Delta\vec{r} &= (\Delta x, \Delta y, \Delta z) \\ \vec{r_i} &= \vec{r_{i-1}} + \Delta\vec{r} \end{aligned} \tag{3.3}$$

There is no spatial limit to the simulation and, as said before, particles that reach the receiver are removed from the simulation. This method is a Martingale process, as a certain step is unrelated to the previous ones.

If flow is added, its contribution should be added to the displacement vector:

$$\begin{aligned} \Delta \vec{r_{flow}} &= (\Delta x, \Delta y, \Delta z) + \delta t * \vec{v_{flow}} \\ \vec{r_i} &= \vec{r_{i-1}} + \Delta \vec{r_{flow}} \end{aligned} \tag{3.4}$$

where $\vec{v_{flow}}$ is the flow velocity vector. This velocity can be a variable of time and space, but for the current project it will be taken as a constant.

The signal obtained is the number of molecules arriving at the receptor at each time, and, therefore, has integers values. Due to the randomness of the process, the final signal presents some noise.

This simulator is used to verify the proposed models, the voxel based simulators.

3.1.2 Voxel-based simulators

The environment is divided into voxels (square or cubes, depending on the dimension) under the assumption that molecules are uniformly distributed inside each voxel.

At each time step, for each voxel, the probability of staying in that voxel is evaluated, and the number of molecules that leave the voxel are distributed to the neighbors. Two versions have been made, depending on the number of neighbors the particles are able to move to at every step:

- **Without diagonals:** Only the direct neighbors are used, 4 in 2D and 6 in 3D, which are; in 2D, the voxels to the *right*, *left*, *up* and *down*, (nh_2, nh_4, nh_1 and nh_3 in Fig. 3.2, respectively), and in 3D the voxels at the *back* and at *front* are added.

- **With diagonals:** Besides the voxels used in the previous version, also the diagonals, and corners in 3D, are added, this is; in 2D, the voxels at the *right-up*, *right-down*, *left-down* and *left-up* (nh_5, nh_6, nh_7 and nh_8 in Fig. 3.2, respectively), and in 3D also the voxels to the back and to the front voxels of this ones, which are direct diagonal and corner voxels.

At every time step, the probability of a transition, P_{trans}, is calculated. In the version with only the direct neighbors are taken into account, in 2D, this is (in 3D is equivalent):

$$P^{2D}_{trans}(nh_i) = \frac{1 - P^{2D}_{stay}}{4} \text{ for } i = 1, 2, 3, 4 \tag{3.5}$$

with nh_i a neighbor voxel (in Fig. 3.2) and P^{2D}_{stay} the probability of staying in the initial voxel (that will be further explained and calculated in a later

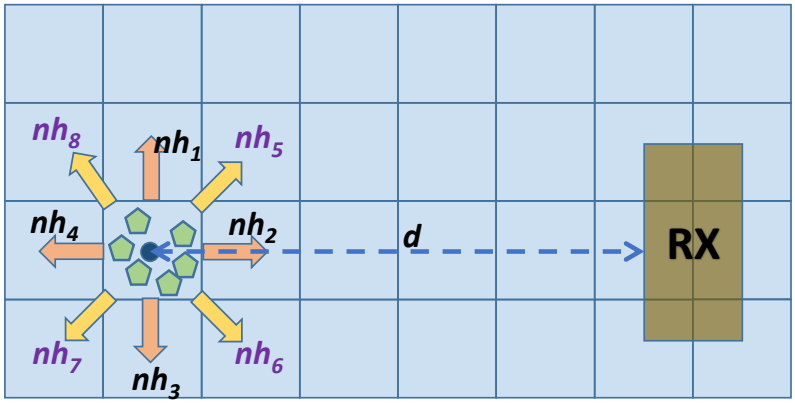


Figure 3.2: Neighbors voxels in a 2D MCvD topology.

section). It is divided by 4 as particles are equally distributed between the 4 neighbors: in 3D, the constant dividing would be 6.

Taking into account the diagonal voxels, these are at a different distance from the center of the initial voxel than the direct ones, and therefore two possibilities arise for this probability of transition:

$$
\begin{aligned}
P^{2D}_{trans}(nh_i) &= \begin{cases} \alpha_1 \left(1 - P^{2D}_{stay}\right) & \text{for } i = 1,2,3,4 \\ \alpha_2 \left(1 - P^{2D}_{stay}\right) & \text{for } i = 5,6,7,8 \end{cases} \\
\alpha_1 &= \frac{1}{4} \frac{P^{2D}_{direct}}{P^{2D}_{direct} + P^{2D}_{diagonal}} \\
\alpha_2 &= \frac{1}{4} \frac{P^{2D}_{diagonal}}{P^{2D}_{direct} + P^{2D}_{diagonal}}
\end{aligned}
\tag{3.6}
$$

where P^{2D}_{direct} and $P^{2D}_{diagonal}$ are the probabilities of transition to a direct neighbor and to a diagonal one, respectively. The first possible probability, for i = 1,2,3,4 is for direct neighbors, while the second one, i = 5,6,7,8 for the diagonal ones (Fig. 3.2) . In the 3D version also the voxels at the corner, forming a 3x3x3 cube, may be taken into account, resulting in three cases for the probability of transition, each with its respective α_i, with i = 1,2,3, that are equivalent to the 2D version.

If flow is also taken into account, the probabilities of transition to different neighbors will vary; for example, for a flow going in the positive x-axis direction, to the right, the probability of transitioning to the right neighbor, P^{2D}_{right}, should be higher than to the left neighbor, P^{2D}_{left}, since the flow drags particles with it in that direction, and therefore there are no general P^{2D}_{direct} or $P^{2D}_{diagonal}$. The flow version of the voxel based in this project does not take into account diagonal neighbors (this would be justified later in the results). For a general flow, that can go to any direction, the probability of transitions would be:

$$
\begin{aligned}
P^{2D}_{trans}(nh_i) &= \begin{cases} \alpha_1 \left(1 - P^{2D}_{stay}\right) & \text{for } i = 1 \\ \alpha_2 \left(1 - P^{2D}_{stay}\right) & \text{for } i = 2 \\ \alpha_3 \left(1 - P^{2D}_{stay}\right) & \text{for } i = 3 \\ \alpha_4 \left(1 - P^{2D}_{stay}\right) & \text{for } i = 4 \end{cases} \\
\alpha_1 &= \frac{P^{2D}_{up}}{P^{2D}_{up} + P^{2D}_{down} + P^{2D}_{right} + P^{2D}_{left}} \\
\alpha_2 &= \frac{P^{2D}_{right}}{P^{2D}_{up} + P^{2D}_{down} + P^{2D}_{right} + P^{2D}_{left}} \\
\alpha_3 &= \frac{P^{2D}_{down}}{P^{2D}_{up} + P^{2D}_{down} + P^{2D}_{right} + P^{2D}_{left}} \\
\alpha_4 &= \frac{P^{2D}_{left}}{P^{2D}_{up} + P^{2D}_{down} + P^{2D}_{right} + P^{2D}_{left}}
\end{aligned}
\tag{3.7}
$$

The new number of molecules at each voxel after a time step becomes, for a certain voxel i:

$$
N^i_{t_{j+1}} = P^{2D}_{stay} N^i_{t_j} + \sum_{k=1}^{k=neighbors} P^{2D}_{trans}(nh_i) N^k_{t_j} \tag{3.8}
$$

The two contributions correspond to the number of molecules that stay in the voxel and the molecules that, have not left the neighbors voxels, diffuse to this voxel.

At every time step, after the movement of the molecules, in the voxels that contain the receptor, the number of molecules proportional to the occupied area are removed from the environment, and this same number constitutes the signal in that time step. On the contrary that with the particle-based solver, in this case the signal of arriving molecules at the receptor is a continuous line, with no noise and taking real values.

Time complexity of this solver is dependent on the number of voxels, independent of the number of molecules, which allows for very high number of molecules without an increase in the runtime. This is its main advantage for big MCvD systems.

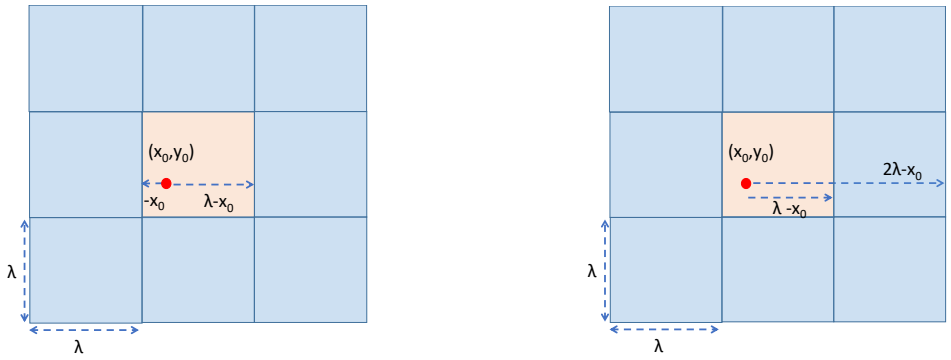
The length of the voxel sides (λ) must be optimized to obtain the correct signal: The optimization of this parameter, depending on the rest of fixed environment parameters in a certain simulation, will be one of the main focuses of this project. Particle-based simulations are used as the correct signal and for a comparison to the signals from voxel-based simulations.

3.2 Implementation

3.2.1 Voxel Probabilities

3.2.1.1 Probability of stay

To calculate the probability of a certain particle, starting in a certain voxel, to stay there or travel to one of its neighbor voxels, in an amount of time δt, it is assumed that the displacement lays within a spatial normal distribution.



(a) Displacement in x for staying in the initial voxel. **(b)** Displacement in x for moving to the right neighbor.

Figure 3.3: From the initial voxel, displacement needed, in the x-axis, to stay in it or move to the right voxel. In the y-axis it will be equivalent, adapted to each probability.

In 2D, the probability of stay, P^{2D}_{stay}, is (from Fig. 3.3a):

$$\begin{aligned} P^{2D}_{stay} &= P(-x_0 \leqslant \Delta x \leqslant \lambda - x_0) \text{ and } P(-y_0 \leqslant \Delta y \leqslant \lambda - y_0) \\ P(-x_0 \leqslant \Delta x \leqslant \lambda - x_0) &= \int_{-x_0}^{\lambda - x_0} \frac{1}{\sqrt{4\pi D \delta t}} e^{\frac{x^2}{4D\delta t}} dx \end{aligned} \tag{3.9}$$

Which can be derived from the normal cumulative distribution function, which is:

$$\int_0^{x_0} \frac{1}{\sqrt{4\pi D \delta t}} e^{\frac{x^2}{4D\delta t}} dx = \frac{1}{2}\left[1 + erf\left(\frac{x}{\sqrt{4D\delta t}}\right)\right] \tag{3.10}$$

with the error function $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$.

$$\begin{aligned} &P(-x \leqslant \Delta x \leqslant \lambda - x) = \\ &= \frac{1}{2}\left[1 + erf\left(\frac{\lambda - x}{\sqrt{4D\delta t}}\right) - 1 - erf\left(\frac{-x}{\sqrt{4D\delta t}}\right)\right] = \\ &= \frac{1}{2}\left[erf\left(\frac{\lambda - x}{\sqrt{4D\delta t}}\right) - erf\left(\frac{-x}{\sqrt{4D\delta t}}\right)\right] \end{aligned} \tag{3.11}$$

From where the probability of staying in the initial voxel can be derived as:

$$P^{2D}_{stay} = \frac{1}{\lambda^2}\int_0^\lambda \int_0^\lambda P(-x_0 \leqslant \Delta x \leqslant \lambda - x_0)P(-y_0 \leqslant \Delta y \leqslant \lambda - y_0)dx_0dy_0 \tag{3.12}$$

3.2.1.2 Neighbor probabilities

To calculate the probability of the particle moving to a neighbor voxel, the limits are changed. For example, to calculate the probability of moving to the right neighbor (from Fig. 3.3b):

$$\begin{aligned} P^{2D}_{right} &= P(\lambda - x_0 \leqslant \Delta x \leqslant 2\lambda - x_0) \text{ and } P(-y_0 \leqslant \Delta y \leqslant \lambda - y_0) = \\ &= \frac{1}{\lambda^2}\int_0^\lambda \int_0^\lambda P(\lambda - x_0 \leqslant \Delta x \leqslant 2\lambda - x_0)P(-y_0 \leqslant \Delta y \leqslant \lambda - y_0)dx_0dy_0 \\ P(\lambda - x_0 \leqslant \Delta x \leqslant 2\lambda - x_0) &= \frac{1}{2}\left[erf\left(\frac{2\lambda - x}{\sqrt{4D\delta t}}\right) - erf\left(\frac{\lambda - x}{\sqrt{4D\delta t}}\right)\right] \end{aligned} \tag{3.13}$$

P^{2D}_{left}, P^{2D}_{up} and P^{2D}_{down} have all the same value, P^{2D}_{direct}.

This is evaluated numerically for the simulations, using (3.11).

In 3D this is equivalent, taking into account also depth direction, and making the integral triple. In this case, there are three different probabilities P^{3D}_{direct}, $P^{3D}_{diagonal}$ and P^{3D}_{corner}, as mentioned before.

For the diagonal neighbors, the calculations are equivalent, only changing the limits to the displacement. In 2D, all the four diagonals will have the same probability, $P^{2D}_{diagonal}$: As an example, the probability for moving to the diagonal up-right neighbor would be:

$$\begin{aligned} P^{2D}_{diagonal} &= P^{2D}_{up-right} = \\ &= P(\lambda - x_0 \leqslant \Delta x \leqslant 2\lambda - x_0) \text{ and } P(\lambda - y_0 \leqslant \Delta y \leqslant 2\lambda - y_0) = \\ &= \frac{1}{\lambda^2}\int_0^\lambda \int_0^\lambda P(\lambda - x_0 \leqslant \Delta x \leqslant 2\lambda - x_0)P(\lambda - y_0 \leqslant \Delta y \leqslant 2\lambda - y_0)dx_0dy_0 \end{aligned} \tag{3.14}$$

If there is flow, probability changes, as the displacement is affected by the flow, as seen in (3.4). If the flow velocity, as assumed for this project, is a constant, and therefore independent of space, the probability changes as:

$$P(\lambda - x_0 \leqslant \Delta x_{flow} \leqslant 2\lambda - x_0) = \delta t v x_{flow} + \frac{1}{2}\left[erf\left(\frac{2\lambda - x}{\sqrt{4D\delta t}}\right) - erf\left(\frac{\lambda - x}{\sqrt{4D\delta t}}\right)\right] \tag{3.15}$$

with vx_{flow} is the flow velocity in the x direction. The probability for displacements in the y-direction will also have a contribution from the flow,

and the product of both probabilities will be integrated as before. This will change depending on the direction of the displacement, as the flow velocity may have different values in different directions. This leads to different probabilities of traveling to different neighbor voxels.

3.2.2 Software

All the software has been developed in Matlab. Due to the complexity of the simulators, it has been split into different .m files, corresponding to the generation of the simulation environment or topology, the different codes for the simulations (with and without diagonal neighbors), and variations on them, both for 2D and 3D.

The data obtained has been analyzed also in Matlab, comparing particle based simulations to voxel based to optimize the latest.

The complete software needed for the simulations appears in the Appendix A. It has been divided in **Topology Generator**, Appendix A.1 and **Simulator**, Appendix A.2. These are the codes in 2D, which will be the ones explained in the following sections, since 3D codes are equivalent, adapted to an extra spatial coordinate.

The units for all the codes in this software are μm for the lengths and $seconds, s$, for the time.

3.2.2.1 Topology Generator

The topology of the environment is created, and different parameters are fixed, related to the environment itself, the transmitter, the receiver, and the simulation.

For the voxel based, a grid of variable size is created; the size of this grid's squares can be chosen, this size being the **voxel length,** λ, and the total length of the environment can be fixed to. Each square of the grid represents a voxel in which the environment is divided, and which the molecules may fill and diffuse from. Another parameter related to the environment fixed here is the **diffusion coefficient, D**.

Positions and size of the transmitter and receiver are also defined here. As said before, the transmitter is assumed to be punctual and in the center of the simulation environment (0 0). The transmitter is put at a certain distance of it, at (d 0), **d** being the distance in the x-axis.

The intersection area of the transmitter and the voxels in the grid is calculated and the area ratios saved with the positions of its correspondent voxels. This is saved in *voxel_rx*, and will be later used for obtaining the signal: An example of it is depicted at Fig. 3.4.

As for the simulation parameters, they are the **time step,** δt, **number of molecules, N** and **time of simulation,** T_{end}.

All the parameters are introduced in the script *board_generate_topology*, Appendix A.1.1: This calls to *hby_prepare2D_sim_voxels*, Appendix A.1.2, which, depending on the parameters introduced, creates the grid and the matrix of the initial state for it, *curr_voxel_set_state*, with all the molecules in the central

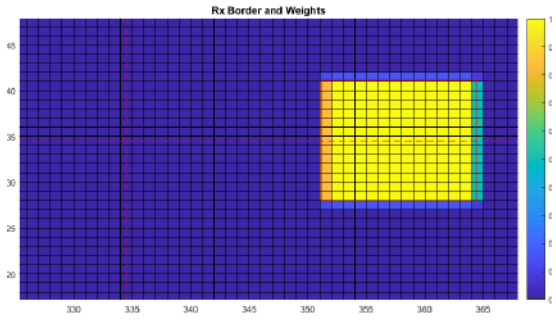


Figure 3.4: Grid near the center of the MC environment, with the axis in red, and receiver (red square). The voxels near the receiver are colored depending on the intersection area with it.

voxel. This state is a representation of the molecules in the environment with which operations can be made, and diffusion will happen in it. This script also calculates the intersection areas explained above. As auxiliary scripts it calls to *evl_coords_to_2Dvoxels_idx*, Appendix A.1.3, which transforms coordinates of the form $[x\ y]$ in points in the state matrix, or voxels in the environment, and *evl_voxel_idx_to_2Dcenter_coords*, Appendix A.1.4, which does the opposite thing, transforms points in the matrix state to usual coordinates.

3.2.2.2 Simulators

hby_simulate2D_particles , Appendix A.2.1 is used as the particle based solver. Since it tracks each particle, and does not use the grid, the transmitter parameters (position and size) are used to obtain its limits in the coordinates, that delimit the perimeter of the transmitter (in 2D, this would be maximum and minimum x and maximum and minimum y). Particle positions are represented by a $2xN$ matrix, N being the number of particles, with the two columns representing the two spatial coordinates, and each row an individual particle.

It initializes with all particles at the origin, and at every time step generates a Gaussian random displacement in another $2xN$ matrix, as seen before in (3.3), each representing the displacement of one particle. After moving them, using the limits of the parameters obtain before, it is evaluated if the new positions of the particles lay inside of the receiver and, if so, the number of them is added to the signal (*hit_timeline*) and they are removed from the environment, causing the matrix containing the positions of the particles, initially of size $2xN$, to decrease. An example of the trajectory for one particle, in this case that does not reach the receiver, for a simulation with $D = 100$, $d = 7$, $T_{end} = 2$ and $\delta t = 10^{-4}$ can be seen in Fig. 3.5.

This kind of trajectory is typical for the Brownian motion. Even though the evident randomness of the motion, there is a total displacement in space, as the final point is separated from the initial one. This would happen for all the particles in every direction.

The voxel based simulator, not taking into account diagonal neighbours, *hby_simulate2D_voxels*, Appendix A.2.2, uses the matrices generated with the

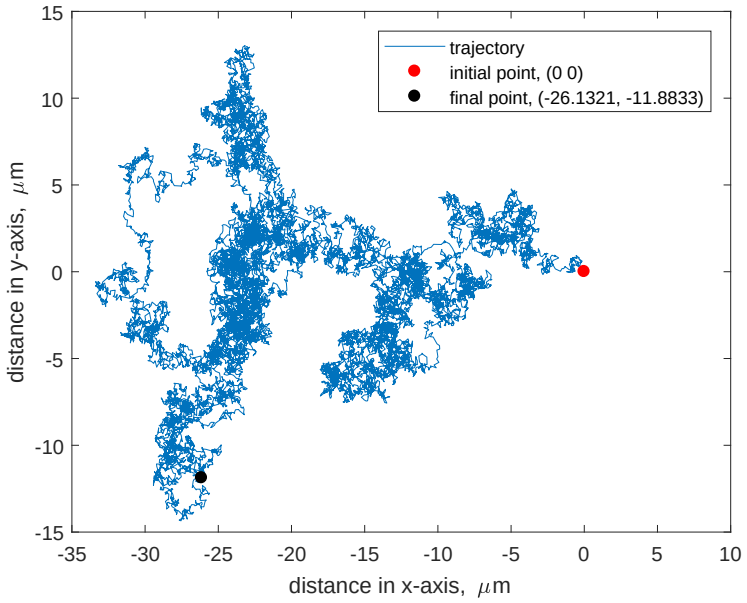


Figure 3.5: Trajectory of one particle for the particle based simulator.

topology, *curr_voxel_set_state*, of the same size as the grid, as the current state of the environment in which it stores the number of molecules in each voxel.

The probability of a certain molecule to stay in its initial voxel, named *voxel_reside_prob*, is evaluated by *evl_pstay2D*, Appendix A.2.6; this is made by the numerical integration defined in (3.12). With it, also the probability of leaving can be evaluated, *voxel_leave_prob*, as $1 - voxel\_reside\_prob$. All this is calculated before beginning the loop, as this values remain constant.

Once inside the loop, at each time step, δt, the state matrix is divided into two matrices, *voxel_set_after_exit* and *voxel_set_exit_molecules*, corresponding to the staying molecules and the leaving molecules, respectively, using the probabilities defined before.

To simulate the diffusing particles moving from each voxel to its direct neighbors, the leaving particle states, *voxel_set_exit_molecules*, is shifted in different directions, corresponding to the directions towards the neighbors, leaving, in 2D, with four shifted versions, up, down, right and left. Since particles are assumed to distribute themselves equally in every direction, having 4 neighbors receiving the leaving particles, these shifted matrices are divided by 4. Doing this, with each shift a column or a row is lost, since the last column of row of the side towards the shift falls out of the environment, and therefore one column or row of zeros needs to be added in the other side. The entire environment is supposed to be big enough so this doesn't affect significantly the signal.

After this, the four shifted matrices and the one with the remaining molecules, *voxel_set_after_exit*, are added. This represents a step in the diffusion.

The scheme for this, for a simple matrix of $5x5$ with initially 5 voxels occupied by molecules, is depicted in Fig. 3.6.

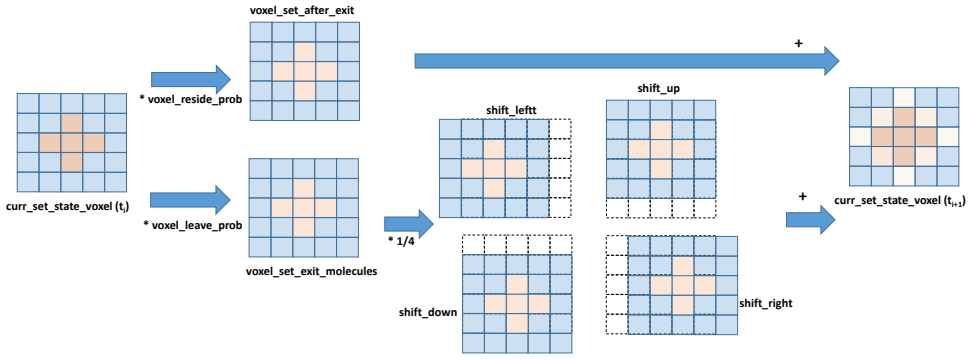


Figure 3.6: One step of voxel based diffusion, only diffusing to direct neighbors.

After every diffusion, the number of molecules that reach the receiver are evaluated with the intersecting areas in *voxel_rx*, and in each voxel reaching molecules are evaluated proportionally to the are ratio. The total number of molecules reaching at a certain time step will constitute the signal at that time step, *hit_timeline(t)*. This molecules are removed from the environment matrix, *curr_voxel_set_state*.

Taking into account diagonals, *hby_simulate2D_voxels_diagonal* Appendix A.2.3, the diffusion is similar, only the different probabilities for direct or diagonal neighbors should be taken into account. The different probabilities are evaluated using *evl_pstay2D_diag* Appendix A.2.7. In this case leaving molecules are not equally distributed in every neighbor voxel: Two constants, *voxel_aligned_prob* and *voxel_diagonal_prob*, are calculated with each probability, defined as in (3.6). Besides the previous shifted states, that are multiplied by *voxel_aligned_prob*, also the diagonal shifted states are calculated, multiplied by *voxel_diagonal_prob*, and added to the previous ones. After this, the rest of the simulation remains the same.

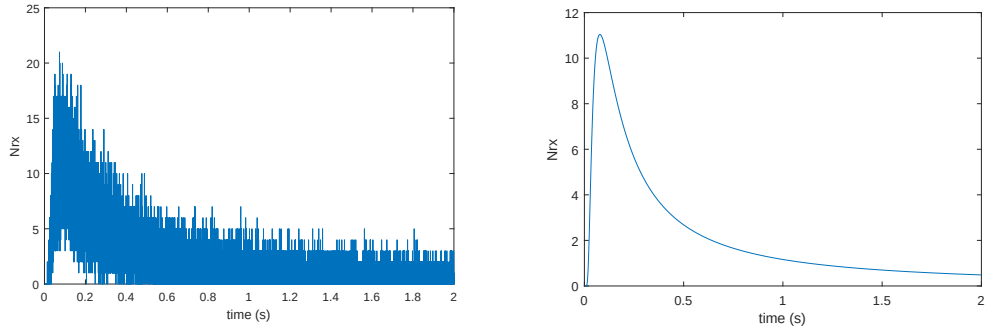
3.2.2.3 Adding flow

If a flow is added to the simulation, for particle based simulations, called *hby_simulate2D_particles_flow*, Appendix A.2.4, the flow contribution is directly added to the displacement in each time step, as in (3.4).

For voxel based, *hby_simulate2D_voxels_flow*, Appendix A.2.5, the diffusive molecules at a certain time step have different probabilities for different directions, and therefore the leaving molecules from a voxel do not distribute themselves equally between all neigbors. The code for the different probabilities are evaluated in *eval_pstay2D_flow*, Appendix A.2.8 , using (3.15), and the leaving molecules, this is, the shifted matrices, are, in each direction, proportional to the probability of that direction, as appears in (3.7).

3.2.2.4 Example of one run

The type of signals obtained by this software, for particle and voxel based (without diagonals) appear in Fig. 3.7.



(a) Received signal versus time for particle based.

(b) Received signal versus time for voxel based.

Figure 3.7: Runs of a) particle based and b) voxel based solvers, for $D = 100$, $d = 7$, $dt = 10^{-4}$, $T_{end} = 2$, $N = 10^5$ and voxel length, $\lambda = 0.2338$.

The differences between both approaches are clear, as particle based generates a signal with a lot of stochastic noise, due to its random behavior, and only of integer values, while voxel based signal arriving at the receiver is a continuous function. In an optimum run, with voxel based simulator optimized, this signal should appear a version with no noise of the particle based.

During this project, particle based signals will be the average of many signals in order to minimize the noise, typically 25, 50 or 100.

3.3 Obtain data

All the simulations of this project have been run using Matlab, in a i9 computer. Due to the complexity of the simulations and the number of it necessary to analyze certain aspects, the runtime for voxel based simulations was very long, specially in 3D, as the increase in the voxels, cubes in this case, was quadratic respect to 2D solvers.

For all simulations T_{end} has been fixed at 2s as the peak was observed to happen around $0.2 \sim 0.5$s, and therefore a total time of 2s was enough for the signal to decay.

Being the main objective of the project to optimize the voxel based simulations, different voxel lengths, λ^{trial}, were tried for each set of parameters and for the variations in them, in order to, using the particle-tracking signals, averaged, compare them and obtain the optimum, λ_{OPT}. The same was done for simulations adding flows of different directions and absolute values.

The number of simulations needed may reach 200 for studying some aspects. In order to shorten as much as possible the runtime of the simulations, when they allowed for it, parallel computing has been used, using the matlab loop comand *parfor*, which functions similarly to the usual *for* loop, but executes the statements in it on a parallel pool of workers. The number of parallel workers available in the computer used was of 10, therefore allowing to 10 different simulations to run at the same time.

This was done first for the 2D codes, and then for 3D. The latter being a much bigger system to compute, as multidimensional arrays, of dimension 3, were used for the 3D grid, as the state of the system, the runtime increased considerably, and the number of parameter values had to decrease in order to have acceptable results in the tests.

3.3.1 Changes in the parameters

During this project it has been analyzed the changes in λ_{OPT} under changes in the diffusion coefficient, **D**, the distance to the receiver, **d**, and the time step, δt.

The motivation under choosing this change of the parameters lays under the form of the Fick's second law (2.3) which, in his discretized version and rearranging can be written as:

$$\begin{aligned} \frac{\Delta C}{\delta t} &= D\Delta C^2 \\ \Delta C &= \frac{1}{\delta t D} \end{aligned} \tag{3.16}$$

where ΔC is the increment in the concentration in a time step δt, as the discretization of the derivation appearing in the original equation. The inverse of the product of δt and D has a linear relation with the increment in the concentration.

Both neighbor models, considering and not considering diagonal neighbors, were used, to analyze if there is a significant improvement using diagonal neighbors, and if there is some change in the λ_{OPT} of both models for the same simulation parameters.

It is important also to consider the runtimes of both models for the λ_{OPT}, which will be discussed later.

3.4 Analysis of the data

Particle-based simulations were compared to different realizations of voxel-based, in order to obtain which of the voxel lengths of the last fits better with the particle based. Two different tests will be used for this comparison; **Normalized Mean Square Error (NMSE)** and **Kolmogorov-Smirnov test (KS test)**.

3.4.1 NMSE

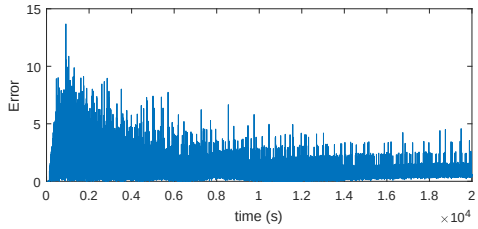
From two realizations, one of the particle-based simulator and the other of the voxel-based, the Normalized Mean Square Error between the signals, *hit_timeline*, of both populations is:

$$NMSE = \frac{||hit\_timeline\_voxel - hit\_timeline\_particle||^2}{||hit\_timeline\_voxel|| * ||hit\_timeline\_particle||} \tag{3.17}$$

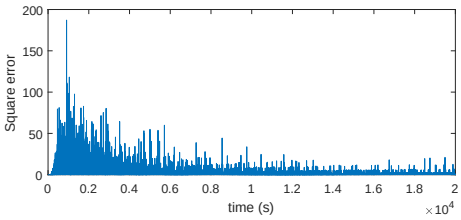
hit_timeline_voxel and *hit_timeline_particle* being the received signal from the voxel-based and particle-based simulators, respectively.

It is of our interest to have a better fit in the peak of the signal than in the tail, since it constitutes the part of the signal with which we may work to encode the information. In practical MCvD systems, the tail is the remaining of molecules after sending a symbol of information, and these remaining molecules affect the next symbol since they remain in the environment; this has been studied and techniques have been developed in order to avoid this effect [10].

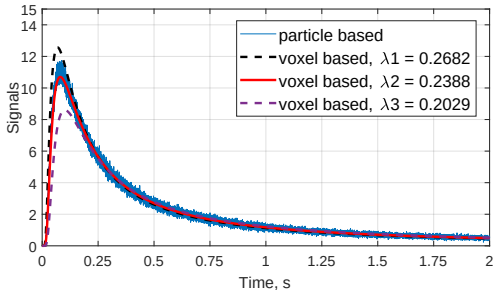
The signal in the receiver has more noise in the peak than in the tail. This can be seen doing the difference between a single realization and the average of many realizations, which will smooth the noise, as in Fig. 3.8a, where the average has been done with 100 realizations. It can be seen how the absolute value of this difference starts growing with the initial time, has its maximum around time 0.1, and then decreases. This maximum coincides with the peak.



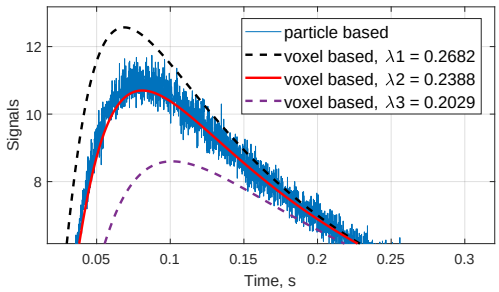
(a) Different between one realization particle based and the average of 100 realizations.



(b) The square of the difference between one realization particle based and the average of 100 realizations.



(c) Received signal for one particle-tracking and three voxel-tracking, with different λs, in 2D environment ($d = 7\mu m$, $D = 100\mu m^2/s$).



(d) Zoom in the peak of the received signals: It is clear that $\lambda 2$ fits it, as the other two fail to.

Figure 3.8

In (3.17), this difference appears to the square, meaning it will give more weight to greater differences: The NMSE will have a greater values as both signals differ in the peak, and smaller if they are closer. This increase in the

error around the peak with respect to the tail error due to the square can be seen in Fig. 3.8b.

The square of the difference has probed to be enough to weight the error in order to make the fit to focus more in the peak: Using different methods to inflate this weight, such as multiplying by the signal itself, that, due to its form, makes the difference in the peak higher, and then renormalizing, the results for the λ_{OPT} did not change, neither by multiplying by the signal with certain exponents.

As an example of the NMSE, in Fig. 3.8c and Fig. 3.8d, which is just a zoom of the previous in the are of interest, the peak, appears an averaged particle based simulation and three different realizations of the voxel based simulator for three different voxels. It can be seen how the three of them fit very similarly in the tail, but differ in the peak, one, $\lambda 1$, higher, another, $\lambda 2$, seeming to fit and the last, $\lambda 3$, which is smaller. Looking to the NMSE values of this three realizations:

	NMSE
$\lambda 1$	0.0568
$\lambda 2$	0.0010
$\lambda 3$	0.0490

The NMSE of $\lambda 2$ is small enough to consider it a good fit, while the other two clearly fail, specially compared to the previous value.

Comparision of the NMSE of different realizations, for different λs, will be one of the main tools in this project, as it has been seen that it seems to predict the signals that fit better with the expectations.

3.4.2 KS test

The Kolmogorov-Smirnov test is a **null hypothesis** test for two samples, this hypothesis being that there is no relation between this samples. It will be used to confirm if the voxel-tracking is a good fit for the particle-tracking signal, if the hypothesis is rejected, and, therefore, both samples are related and may come from the same system: this will mean the voxel-tracking simulation is a good enough approximation for the MCvD system, and there is no need for particle based simulations.

The two-sample KS test is used for this project, which can be done by a command in Matlab, *kstest2(...)*. This test determines if two random samples (particle and voxel based signals) are drawn from the same population. It returns two values: **P-value**, which is higher as the samples are more likely to come from the same system, and **H-value**, which is equal to 0 if the null hypothesis is not rejected, this is, if both samples actually may come from the same population, and equal to 1 if the may not. The significance level for the H-value by default is set to 0.05.

For the same simulations as before, in Fig. 3.8c, the KS gives a H-values and P-values as following:

	KS H-value	KS P-value
$\lambda 1$	0	0.0126
$\lambda 2$	0	0.3817
$\lambda 3$	0	0.000128

This is additional to the NMSE values of the previous table, and, in the case of $\lambda 2$, it confirms the goodness of test as its H-value is 0 and its P-value high enough, rejecting the null hypothesis. The other two λ fail the test.

One problem for the used script arose when using the averaged signal, as the original software for the KS test only worked with integers (non zero) for the particle-tracking signal. This problem can be solved by rescaling, using directly the sum of all the signals previously averaged, which are originally integers and so will be their sum, and multiplying the voxel-tracking signal by the number of summed signals, so the test will have the same meaning.

For this project, NMSE values will be used in order to obtain the optimum voxel length value, λ_{OPT}, and KS test as an additional confirmation (or rejection) of this value as a good enough fit.

Chapter 4

Simulation results and discussion

4.1 2D MCvD systems

First, we will consider a 2D MCvD system and analyze under variations of the environment parameters and neighbors model the optimization of the voxel-tracking simulator, using the tools explained in the previous chapter.

Since 3D systems take much longer to simulate, we might use some of the 2D results in order to save simulation time.

4.1.1 Neighbor models

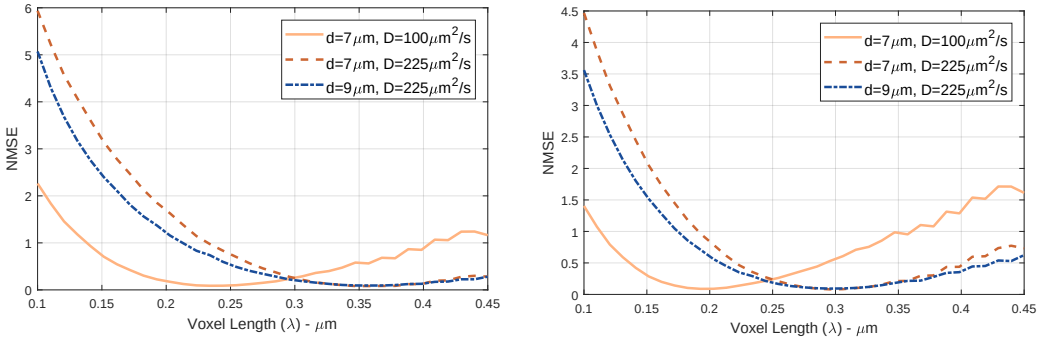
For comparing both neighbor models, the one considering diagonal neighbors as well as the direct ones and the one without them, three different scenarios are run, with different diffusion coefficients, D, and distance to the receiver, d. Simulation parameters appear in Table 4.1: For the voxel length, 35 different values equally spaced between the values listed are tried, λ^{trial}, in order to obtain for which the NMSE between the particle-tracking and voxel-tracking (of this specific λ) is optimum. The particle based signal is one for each scenario and is done by the averaging of 100 realizations. We can see this results in Fig. 4.1.

Table 4.1: Simulation parameters

Parameter	Value
Distance (d)	$\{7, 9\}$ µm
Diffusion coefficient (D)	$\{100, 225\}$, µm^2/s
Simulation time step (δt)	0.0001 s
Number of released molecules (N^{Tx})	100 000
Voxel length (λ^{trial})	$0.1 \sim 0.45$ µm , $\Delta\lambda = 0.01$

It can be clearly seen that the NMSE results, for all the three scenarios in both models has a clear minimum, and it grows as the λ^{trial} go away from it, to higher and lower values. The λ_{OPT} for these simulations appear in Table 4.2. λ_{OPT} values considering diagonal neighbors, λ_{OPT}^{with}, are smaller than in the version without diagonals λ_{OPT}^{wout}.

The KS test for these simulations doesn't give results precise enough, as for the used λs the resolution is not as small as it is needed for this kind of



(a) NMSE plots not considering diagonal neighbors.

(b) NMSE plots considering diagonal neighbors.

Figure 4.1: NMSE for 2D MCvD systems with different parameters, for both neighbor models.

Table 4.2: λ_{OPT} and its NMSE value for both neighbor approaches

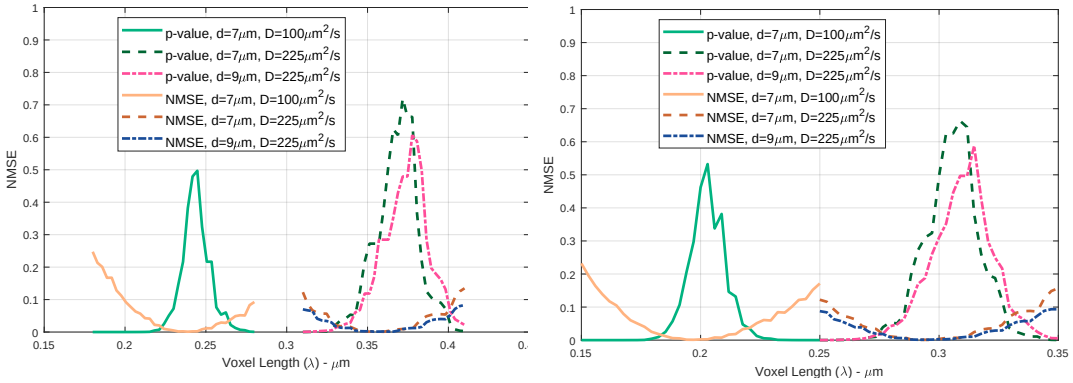
Scenario (D,d) = (μm^2/s,μm)		λ_{OPT}^{NMSE}(μm)	NMSE
Without diagonals	$D = 100, d = 7$	**0.2388**	0.001773
	$D = 225, d = 7$	**0.3574**	0.00124
	$D = 225, d = 9$	**0.3574**	0.00124
With diagonals	$D = 100, d = 7$	**0.1926**	0.002407
	$D = 225, d = 7$	**0.2956**	0.000931
	$D = 225, d = 9$	**0.2956**	0.0009899

tests. Therefore, a second group of simulations were realized, in this case with different λs around the previously obtained λ_{OPT}, with smaller separation between values, and also 35 different values for each separation. These values, for each scenario and model, appear in Table 4.3.

Table 4.3: λ values tried for each scenario.

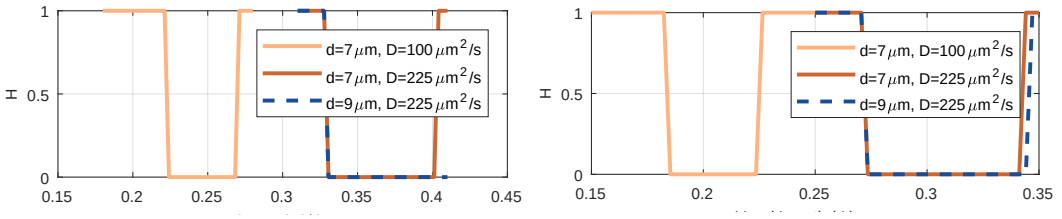
Scenario		Voxel length (λ), $\Delta\lambda = 0.0029$
Without diagonals	$D = 100\mu m^2/s, d = 7\,\mu m$	$0.18 \sim 0.28\,\mu m$
	$D = 225\mu m^2/s, d = 7\,\mu m$	$0.31 \sim 0.41\,\mu m$
	$D = 225\mu m^2/s, d = 9\,\mu m$	$0.31 \sim 0.41\,\mu m$
With diagonals	$D = 100\mu m^2/s, d = 7\,\mu m$	$0.15 \sim 0.25\,\mu m$
	$D = 225\mu m^2/s, d = 7\,\mu m$	$0.25 \sim 0.35\,\mu m$
	$D = 225\mu m^2/s, d = 9\,\mu m$	$0.25 \sim 0.35\,\mu m$

The NMSE and KS test values, in this case both P and H-values, are in Fig. 4.2. P-value and the NMSE, even though its respective values having different meanings, have been represented for each case in the same plots, as from both a single optimum voxel length can be obtained.



(a) NMSE and P-values for 2D system, not considering diagonal neighbors.

(b) NMSE and P-values for 2D system, considering diagonal neighbors.



(c) H-values for 2D system, not considering diagonal neighbors.

(d) H-values for 2D system, considering diagonal neighbors.

Figure 4.2: NMSE and KS test P and H-values for both neighbor models, for the same three scenarios as the previous.

Results for λ_{OPT} from this data is represented in Table 4.4. For the NMSE, the values are similar, slightly corrected respect to the previous version, and in the plots it is also clear that there is one single minimum and as λ values move away from it their correspondent NMSE grow.

The Gaussian total variance for this simulations is $\sqrt{4D\delta t}$, which, for the current simulations, with $\delta t = 1e-04$s has values **0.2** and **0.3** for $D = 100\mu m^2/s$ and $D = 225\mu m^2/s$, respectively. We can see the λ_{OPT} values are close to them, specially for the version with diagonal neighbors.

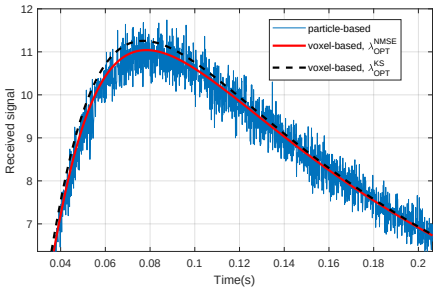
The P-values, which is higher as both samples are more likely to be related, has peaks and a clear higher value, which will give the λ_{OPT}. In the table we can see how for the same scenario and model the λ_{OPT} given by highest P-values are slightly higher that those chosen with the lowest NMSE.

For the H-values, with default significance level giving the KS test as positive for any P-value higher than 0.05, there is a window of values which pass the test, and both λ_{OPT}^{NMSE} and λ_{OPT}^{KS} lay in the acceptance regions, meaning both optimum values are acceptable for this significance level.

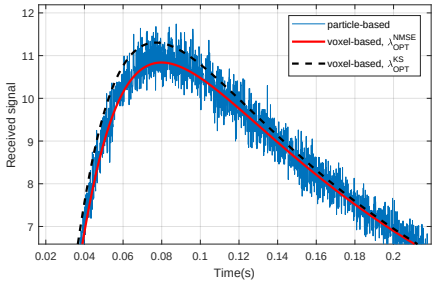
In Fig. 4.3 the peak of the particle-tracking signal and the voxel-tracking signals for λ_{OPT}^{NMSE} and λ_{OPT}^{KS} is represented, for both neighbor models and the three scenarios. As stated before, fitting the peak should be the main focus of this studies. Although it varies between the six plots, it is clear that the λ_{OPT}^{NMSE} fits better to the particle-tracking, as it seems to be similar to a version of it with no noise, while the λ_{OPT}^{KS} fails, being a little bit too high in all cases.

Table 4.4: λ_{OPT} from NMSE and KS tests, for both neighbor approaches

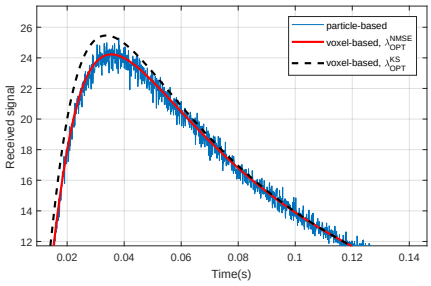
Scenario (D,d) = ($\mu m^2/s$,μm)		$\lambda_{OPT}^{NMSE}(\mu m)$	NMSE	$\lambda_{OPT}^{KS}(\mu m)$	P-value
Without diagonals	$D = 100, d = 7$	**0.2388**	0.001035	**0.2447**	0.4968
	$D = 225, d = 7$	**0.3600**	0.000864	**0.3718**	0.7194
	$D = 225, d = 9$	**0.3541**	0.001125	**0.3776**	0.6064
With diagonals	$D = 100, d = 7$	**0.1971**	0.001075	**0.2029**	0.5325
	$D = 225, d = 7$	**0.2971**	0.000885	**0.3088**	0.6631
	$D = 225, d = 9$	**0.2941**	0.00112	**0.3147**	0.5877



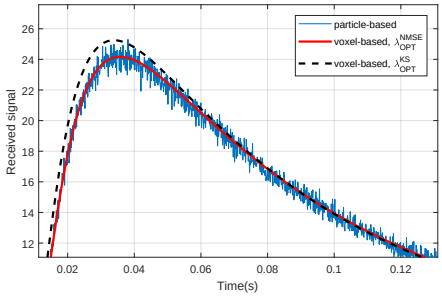
(a) $D = 100\mu m^2/s$, $d = 07\mu m$, not considering diagonal neighbors.



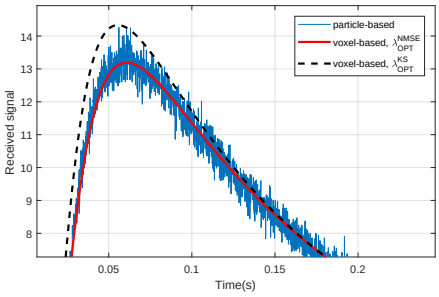
(b) $D = 100\mu m^2/s$, $d = 07\mu m$, considering diagonal neighbors.



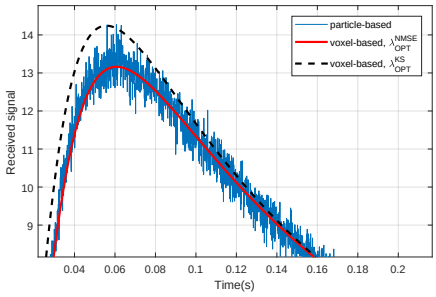
(c) $D = 225\mu m^2/s$, $d = 07\mu m$, not considering diagonal neighbors.



(d) $D = 225\mu m^2/s$, $d = 07\mu m$, considering diagonal neighbors.



(e) $D = 225\mu m^2/s$, $d = 09\mu m$, not considering diagonal neighbors.



(f) $D = 225\mu m^2/s$, $d = 09\mu m$, considering diagonal neighbors.

Figure 4.3: Peaks of the received signal for particle based and voxel based, both λ_{OPT}^{NMSE} and $\lambda_{OPT}P^{KS}$.

In Fig. 4.4 we can see the runtime of different simulations versus the number of particles released during each: The simulations are particle based and voxel based in each neighbor model. These simulations are for a MCvD environment with $D = 100$, $d = 7$ and $\delta t = 1e-04$, and for each voxel neighbor model its λ_{OPT} has been chosen from Table 4.4; **0.2338** for without diagonal neighbors and **0.1971** when considering diagonal neighbors.

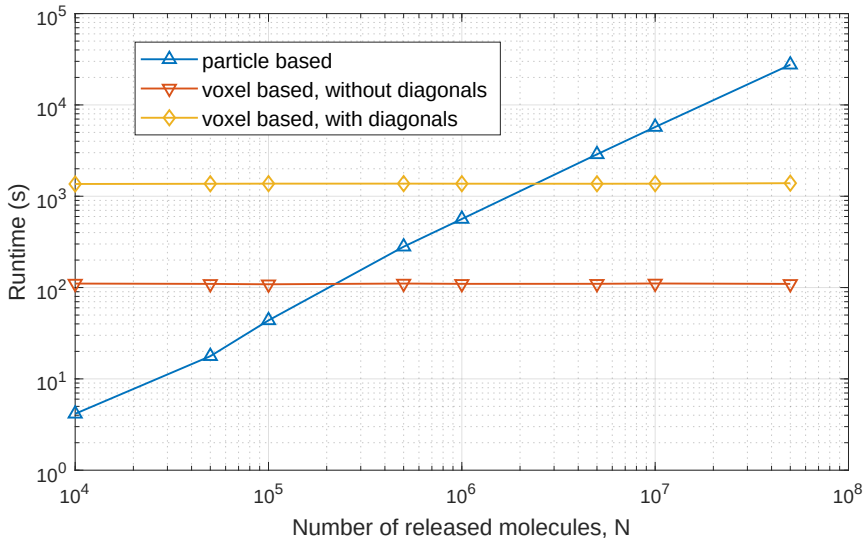


Figure 4.4: Runtimes of three simulators, for MCvD 2D system with $D = 100$, $d = 07$ and $\delta t = 1e-04$.

We can see how, while both voxel models have a constant runtime as the number of released molecules grows, which is consistent with the stated before that this number shouldn't affect, since only the number of particles in each voxel is simulated, and not the particles themselves, the particle based simulator grows with the number of released molecules. Even if for a small number of released molecules particle-tracking simulators takes less time, as this number grows, to a number reasonable and even small for certain MCvD systems, the increase in its runtime, specially compared with both voxel-tracking models, makes it inconvenient to practical use.

Between both neighbor models, for the same number of released molecules an with their respective λ_{OPT}, taking account diagonal neighbors has a significantly larger runtime. This is due to two factors; adding the diagonal contributions means some extra operations and, the most important one, the λ_{OPT}^{with} is smaller than λ_{OPT}^{wout} and, since the total length of the environment is fixed, it has to simulate a greater number of voxels.

4.1.2 Effect of system parameters

From all the simulations above, we can see how λ_{OPT} seems to grow with the diffusion coefficient, and that with changes in the distance its change is much less noticeable, if actually there is some.

To study this in more depth, more simulations were made with varying D and d, in order to see the relation between its change and λ_{OPT}.

The same was done for the time step dt.

4.1.2.1 Variations in D

Simulation parameters appear in Table 4.5. 10 different values for D, between 100 and 400, are selected, and in each of them three different distances to the receiver are simulated.

In order to obtain the $\lambda_{OPT}(D)$, at every different environment, which in this case is a set of $\{D, d\}$, a number of λ^{trial} are simulated in a voxel-tracking simulator, 25 particle-based simulation are run and averaged, and the optimum is chosen by the lowest NMSE test. From this optimum value, the next values for λs for the following D are chosen between some reasonable values, as λ_{OPT} is expected to grow with D.

In order to choose, for every set of parameters, the λ_{OPT}^{NMSE}, no NMSE minimum value above 10^{-2} have been accepted as correct, which lead to different simulations until reaching a satisfactory one, that fulfilled this.

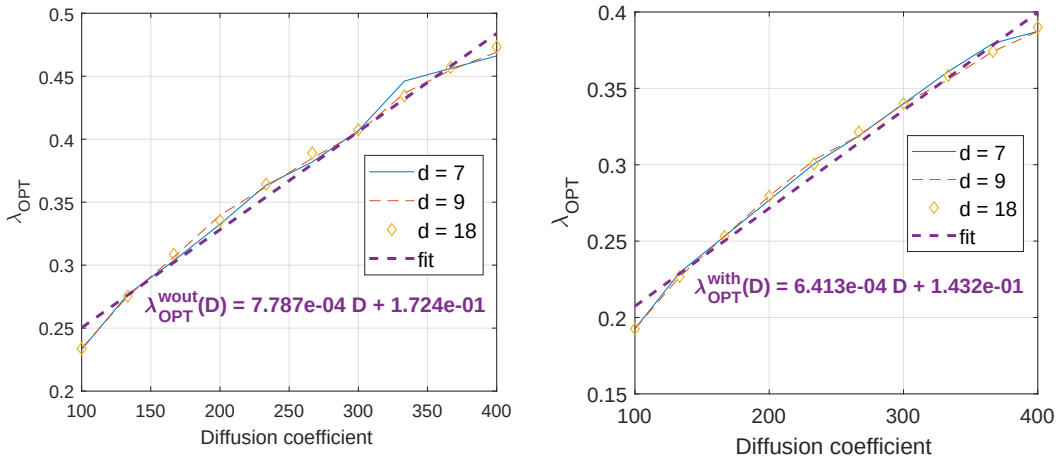
Table 4.5: Simulation parameters

Parameter	Value
Distance (d)	$\{7, 9, 18\}$ µm
Diffusion coefficient (D)	$100 \sim 400$µm^2/s, $\Delta D = 30$
Simulation time step (δt)	0.0001 s
Number of released molecules (N^{Tx})	100 000

It is first noticeable how the effect of D in λ_{OPT} appears to be approximately linear, for both neighbor models. A linear fit can be tried following this idea, shown, with the results, in Fig. 4.5.

Also, from the plots seems that, as far as these results go, the distance does not affect λ_{OPT}, as the variation appears in it while D grows appears to be the same independently of the distance to the receiver, with some differences due to the resolution of λ^{trial} for the simulations.

The optimum voxel lengths obtained for the neighbor model that considers diagonal voxel,$\lambda_{OPT}^{with}(D)$, have a smaller value than those from the model without diagonal neighbors, $\lambda_{OPT}^{wout}(D)$, and the slope that relates them linearly with the diffusion coefficient is also smaller for the $\lambda_{OPT}^{with}(D)$. This is a generalization of the previous results for only three scenarios, that showed that for the same scenario λ_{OPT}^{with} was smaller than λ_{OPT}^{wout}. This may be explained since, not considering diagonals neighbors, particles only reach them in two time steps instead of one, so, in order to travel the same distance going diagonal in both models this difference in voxel length is needed.



(a) Linear fit for λ_{OPT} not considering diagonal neighbors versus D.

(b) Linear fit for λ_{OPT} considering diagonal neighbors versus D.

Figure 4.5: $\lambda_{OPT}(D)$ versus diffusion coefficient linear fit for both neighbor models.

4.1.2.2 Variations in δt

Simulation parameters appear in Table 4.6. As with variations in the diffusion coefficient, three distances have been simulated for each time step.

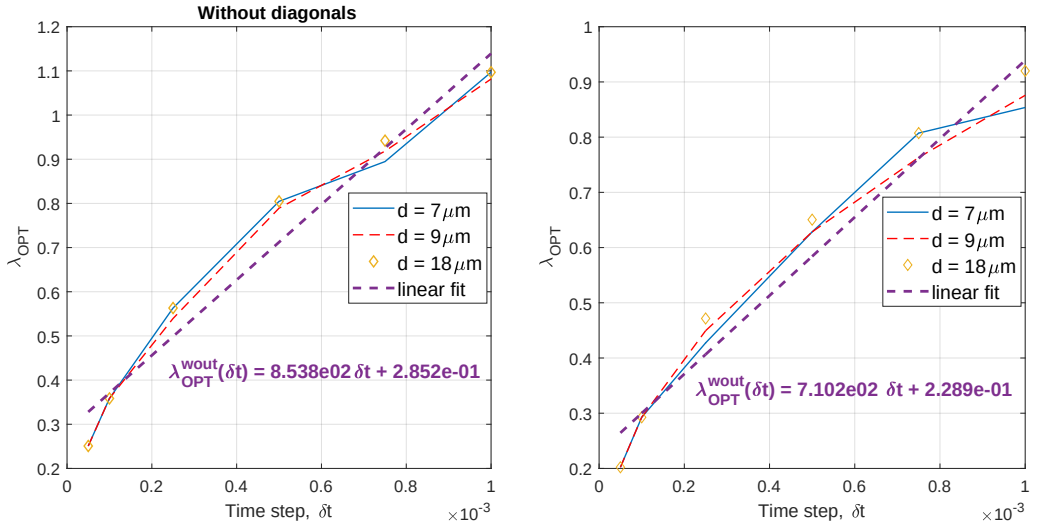
Table 4.6: Simulation parameters

Parameter	Value
Distance (d)	$\{7, 9, 18\}$ µm
Diffusion coefficient (D)	$225 \mu m^2/s$
Simulation time step (δt)	$\{0.05, 0.1, 0.25, 0.5, 0.75, 1.0\}e-03$s
Number of released molecules (N^{Tx})	100 000

Similarly as the previous result, the changes in the distances doe not cause any apparent change in the results, and increasing the time step led to a linear increase in λ_{OPT} in both neighbor models, although a curve might be appreciated in both cases. A linear fit could be performed; these results appear in Fig. 4.6.

Again, $\lambda^{with}_{OPT}(\delta t)$ has smaller values than $\lambda^{wout}_{OPT}(\delta t)$, and the slope of the linear fit is also smaller for the version with diagonal neighbors. It is noticeable that the proportion of this difference is almost the same for both varying parameter simulations: for variations in D, the proportion between the slopes, $\frac{slope^{wout}_{D}}{slope^{with}_{D}}$, is equal to **1.2143**, while for variations in δt, $\frac{slope^{wout}_{\delta t}}{slope^{with}_{\delta t}}$, this is **1.2022**.

The value of these proportions is also similar to that between the voxel lengths of each model for the first simulation, whose values appear in Table 4.4: This proportions are listed in Table 4.7.



(a) Optimal voxel length versus time step, δt, not considering diagonal neighbours, and linear fit.

(b) Optimal voxel length versus time step, δt, considering diagonal neighbours, and linear fit.

Figure 4.6: $\lambda_{OPT}(\delta t)$ versus δt and linear fit for both neighbor models, 2D MCvD system.

This value being consistent within different simulations suggest the relation of λ_{OPT} with variations in D or δt is independent of the neighbor model used, as the linearity is the same for both, and the differences in the slope appearing in the results for both models are due to differences in the λ_{OPT} themselves, and not actually in the linearity.

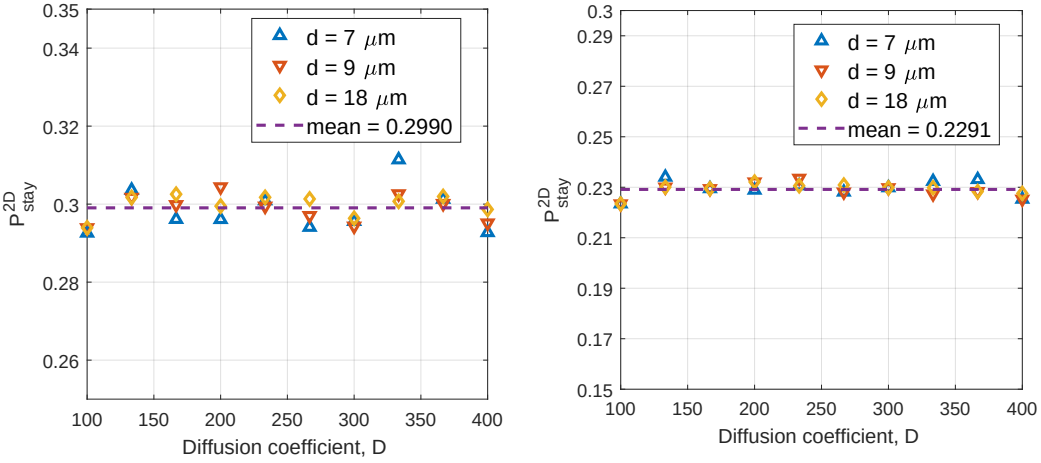
Table 4.7: Proportion between optimum voxel length for both neighbor models

Scenario (D,d) = (μm^2/s,μm)		$\frac{\lambda_{OPT}^{NMSE,wout}}{\lambda_{OPT}^{NMSE,with}}$	$\frac{\lambda_{OPT}^{KS,wout}}{\lambda_{OPT}^{KS,with}}$
Without diagonals	$D = 100, d = 7$	**1.2116**	**1.2060**
	$D = 225, d = 7$	**1.2117**	**1.2040**
	$D = 225, d = 9$	**1.2040**	**1.1999**

4.1.3 Voxel transition probabilities

From the previous results, varying D and dt, the probability of stay, P_{stay}^{2D}, of the optimum voxel length, $P_{stay,OPT}^{2D}$, can be calculated. This appeared not to vary between scenarios, as shown, for variations in D, in Fig. 4.7, and for variations in dt, in Fig. 4.8, both plots with the same simulations parameters as those from the previous sections.

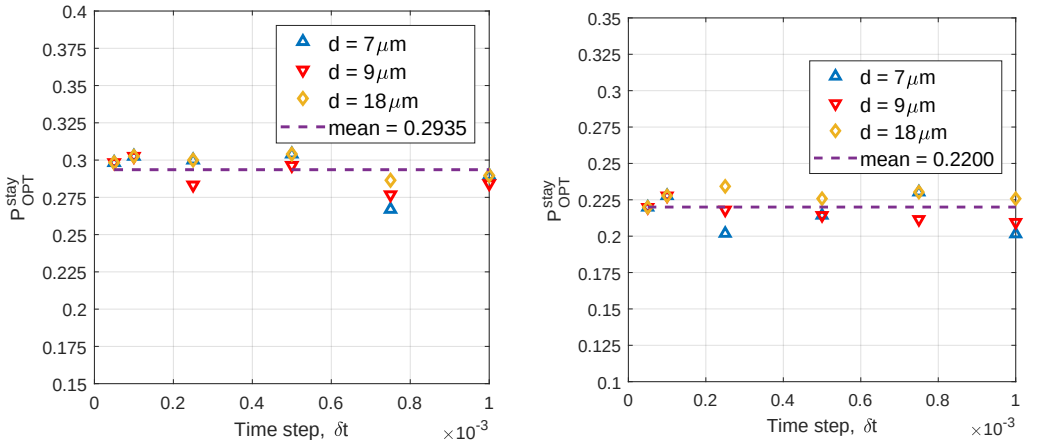
We can see that between the two neighbor models, when diagonal neighbors are considered, $P_{stay,OPT}^{2D}$ has a smaller value that when they are not



(a) Not considering diagonal neighbors.

(b) Considering diagonal neighbors

Figure 4.7: Probability of stay for λ_{OPT} for 2D MCvD system under variations in the diffusion coefficient.



(a) Not considering diagonal neighbors.

(b) Considering diagonal neighbors

Figure 4.8: Probability of stay for λ_{OPT} for 2D MCvD system under variations in the time step.

considered. This is related with the fact, seen in the previous sections, that λ_{OPT}^{with} for a certain set of parameters has a smaller value than λ_{OPT}^{wout}, meaning the central voxel from which this probability is calculated is smaller, and so should the probability of staying in it, for the same parameters, be.

In any both neighbor models, $P_{stay,OPT}^{2D}$ has a very similar mean value for variations in the diffusion coefficient and the time step, which can be considered a constant related with the optimal voxel length.

This constant value fixes a relation between the diffusion coefficient, the time step and the voxel length, since the probability only depends on this three parameters, (3.11) and (3.12). For a certain simulation, if the environment parameters, D, d and δt, are fixed (the voxel length, λ, is not a parameter of the environment itself but only of the voxel-tracking simulator), the value

$P^{2D}_{stay,OPT}$ and its integral definition allows to find directly the λ_{OPT}.

4.2 3D MCvD systems

Based on the 2D results of the previous sections comparing both neighbor models, we see adding diagonals does not have any significant improvement in the NMSE test, and, due to the smaller length of its λ^{with}_{OPT}, its runtime is actually higher than the one of the version that only considers direct neighbors. Also, all the results, varying the diffusion coefficient D, the distance to the receiver d and the time step δt are reproducible in both models, even though having some values different, the behavior remains the same. Due to this, and considering that 3D systems take a much higher runtime than 2D for voxel-tracking simulations, as the number of voxels increase drastically, we will continue only with the voxel based simulator that does not take into account diagonal neighbors, as it does not seem that taking into account them improves the results, while it does consume a lot of computational time.

First, as with 2D environments, simulations for three different 3D MCvD scenarios, with variations in the parameters of each of them, are done, with different voxel length, λ^{trial}. Simulation parameters for this appear in Table 4.8, and the results in Fig. 4.9.

Table 4.8: Simulation parameters

Parameter	Value
Distance (d)	$\{7, 9\}$ µm
Diffusion coefficient (D)	$\{100, 225\}$, µm^2/s
Simulation time step (δt)	0.0001 s
Number of released molecules (N^{Tx})	100 000
Voxel length (λ)	$0.2 \sim 0.5$ µm , $\Delta\lambda = 0.0086$

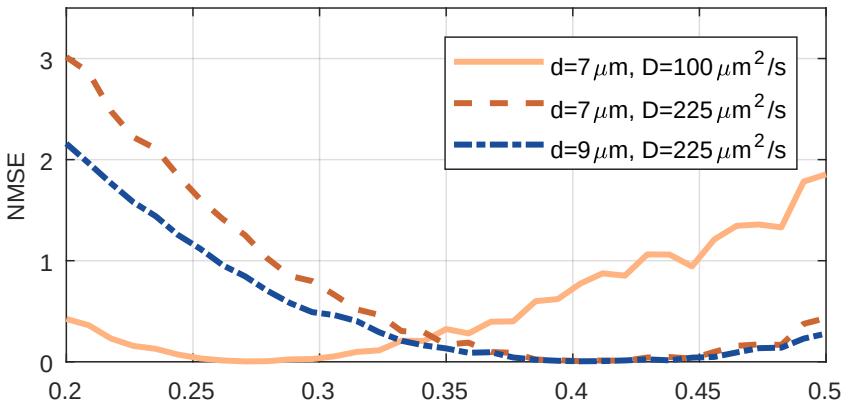


Figure 4.9: NMSE for 3D MCvD three scenarios.

The NMSE plots are similar to those in 2D, but with different values for λ_{OPT}, which are listed in Table 4.9. There is an increase in its value, related with the increase in dimensions. However, as before, in order to obtain more accurate results a second set of simulations, with λ^{trial} around this λ_{OPT} were made. The results for its NMSE and KS test appear in Fig. 4.10, and the value of λ_{OPT}^{NMSE} and λ_{OPT}^{KS}, with the value of the test for them, for each scenario are listed in Table 4.10.

Table 4.9: λ_{OPT} and its NMSE value for 3D McvD environments.

Scenario (D,d) = (μm^2/s,μm)	λ_{OPT}^{NMSE}(μm)	NMSE
$D = 100, d = 7$	**0.2706**	0.003671
$D = 225, d = 7$	**0.4029**	0.006129
$D = 225, d = 9$	**0.4029**	0.006129

Table 4.10: λ_{OPT} from NMSE and KS tests, for 3D MCvD

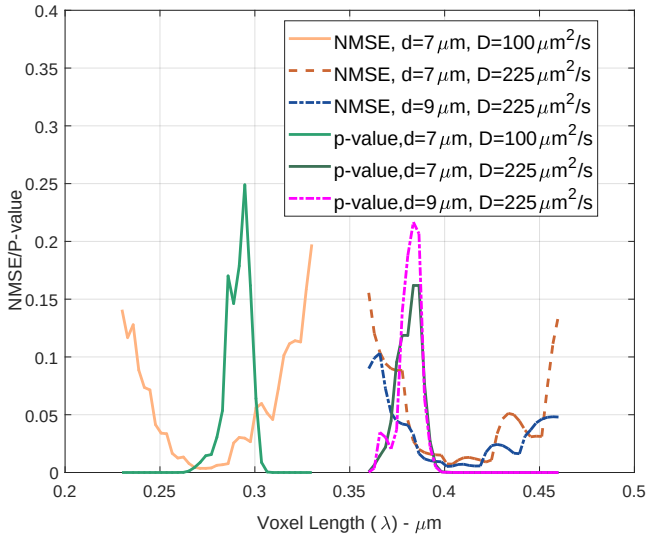
Scenario (D,d) = (μm^2/s,μm)	λ_{OPT}^{NMSE}(μm)	NMSE	λ_{OPT}^{KS}(μm)	P-value
$D = 100, d = 7$	**0.2741**	0.003596	**0.2947**	0.249
$D = 225, d = 7$	**0.4041**	0.007207	**0.3835**	0.1618
$D = 225, d = 9$	**0.4012**	0.005232	**0.3835**	0.2167

It is first noticeable that λ_{OPT} of each scenario is slightly larger than its equivalent in 2D. Again, in 3D the Gaussian total variance has the form $\sqrt{6D\delta t}$, as an extra dimension respect is added respect to 2D. Therefore, this increase should be expected. The value of this variance is, for $D = 100$μm^2/s **0.2449** and for $D = 225$μm^2/s **0.3674**, which again is close to the obtained λ_{OPT}.

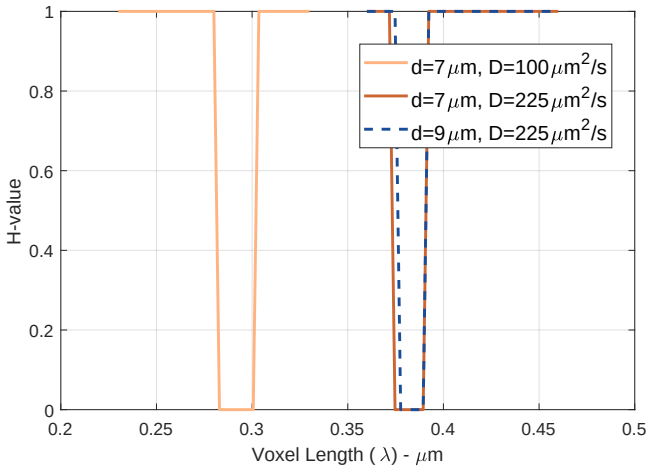
In this case the window of values with a positive result in the KS test is narrower than before, due to the smaller values of the KS test P-values. Also, while for 2D the λ_{OPT}^{KS} had a value larger than λ_{OPT}^{NMSE} in all three scenarios, due to a displacement in the peak of the P-values respect to the minimum of the NMSE, which was always in the same direction, in this case for the two scenarios with $D = 225$μm^2/s this displacement happens in the opposite direction, with the peaks of the P-values in a lower voxel length values than the minimum of the NMSE, causing λ_{OPT}^{KS} to be smaller than λ_{OPT}^{NMSE} for these two cases.

Also, due to the narrowness of the accepted region, the λ_{OPT}^{NMSE} in all three scenarios lay, even though near, out of this region. Changing the significance level, for example to 0.01, the window of accepted values is broader and the λ_{OPT}^{NMSE} are accepted.

The difference in these results, specially compared to those equivalent in 2D, may be due to particles diffusing in more directions, as they have one extra direction to diffuse to, and therefore making the signal to be of less number of molecules for the same released molecules, and cause the KS tests



(a) NMSE and P-value for 3D MCvD three scenarios.



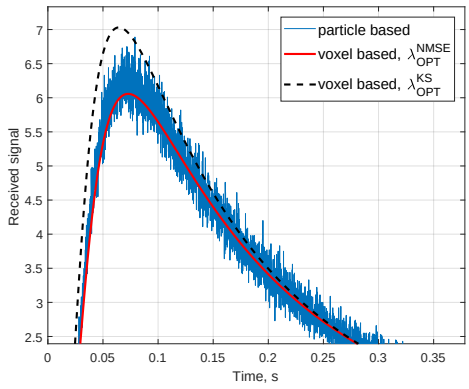
(b) KS test K-value for 3D MCvD three scenarios.

Figure 4.10: Results for λ_{OPT}^{3D}.

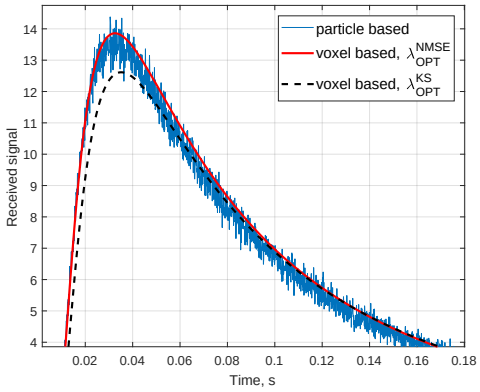
not to be so reliable. To have results as precise as in 2D maybe simulations with more released molecules are needed.

In Fig. 4.11 we can see the received signal in each of the three scenarios studied, for the particle-tracking solver and for the two optimum voxel lengths, λ_{OPT}^{NMSE} and λ_{OPT}^{KS}: The plots are focused in the peak of the signal. λ_{OPT}^{NMSE} seems to fit better, while λ_{OPT}^{KS} fails, having values too high in the first scenario, where its λ_{OPT} was higher than the one from the NMSE test, and being too low in the other two, where its λ_{OPT} was too low.

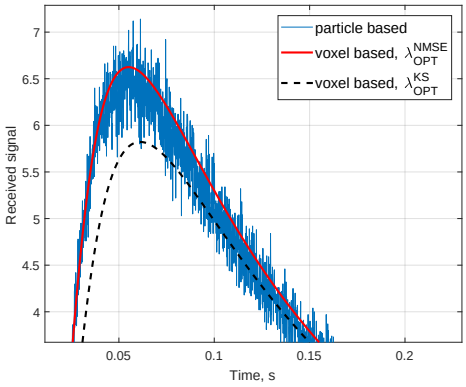
Again, even though the KS test, we are forced to accept as a better fit the values from the NMSE.



(a) $D = 100\mu m^2/s$, $d = 07\mu m$.



(b) $D = 225\mu m^2/s$, $d = 07\mu m$.



(c) $D = 225\mu m^2/s$, $d = 09\mu m$.

Figure 4.11: Particle based compared to voxel based, both λ_{OPT}^{NMSE} and λ_{OPT}^{KS}, for 3D MCvD three scenarios.

4.2.1 Effect of system parameters

Again, simulations varying the simulation parameters D, d and δt are run. Due to the computational time required to do this in 3D, we were forced to reduce the number of different environments in which the λ_{OPT} was looked for, in order to maintain an acceptable resolution between λ^{trial} in every environment and acceptable standards in the NMSE tests.

4.2.1.1 Variations in D

The parameters for the simulation appear in Table 4.11. To ensure NMSE values low enough for ensuring an acceptable λ_{OPT}^{NMSE}, the one used in this simulations, only 4 diffusion coefficients and two distances were simulated.

The results, similarly as in 2D, do not seem to vary with the distance, and the optimum voxel length presents a linear relation with the diffusion coefficient. In Fig. 4.12 we can see this results, and the linear fit. The slope is similar but higher than in 2D, which linear fit for variations in D, not considering, as here, diagonal neighbors, had a slope of $7.787e-04$. This is consistent with the fact that the λ_{OPT}^{3D} had larger values than λ_{OPT}^{2D}.

Table 4.11: Simulation parameters

Parameter	Value
Distance (d)	$\{7, 9\}$ µm
Diffusion coefficient (D)	$100 \sim 400 \mu m^2/s$, $\Delta D = 100$
Simulation time step (δt)	0.0001 s
Number of released molecules (N^{Tx})	100 000

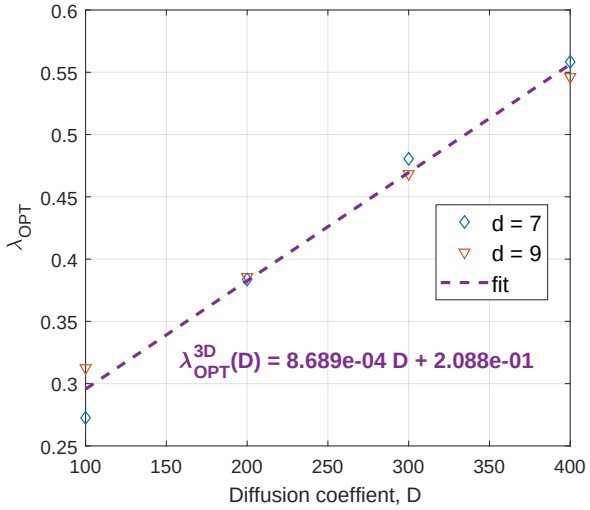


Figure 4.12: $\lambda_{OPT}(D)$ versus the diffusion coefficient and its linear fit in a 3D MCvD system.

4.2.1.2 Variations in δt

Simulation parameters are listed in 4.12. Results for the λ_{OPT} appear in Fig. 4.13: As before, there is no difference between both distances and a linear fit can be tried.

Table 4.12: Simulation parameters

Parameter	Value
Distance (d)	$\{7, 9\}$ µm
Diffusion coefficient (D)	$225 \mu m^2/s$
Simulation time step (δt)	$\{0.05, 0.1, 0.25, 0.5, 0.75, 1.0\}e-03$s
Number of released molecules (N^{Tx})	100 000

As for 2D results, the slope for variations in the time step is orders of magnitude bigger than the one from variations in the diffusion coefficient, and higher than the same slope for the 2D results, which was $8.538e02$ for the model with no diagonal neighbors.

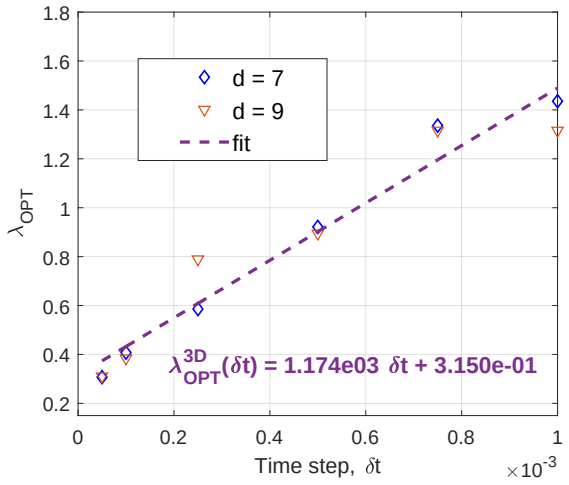
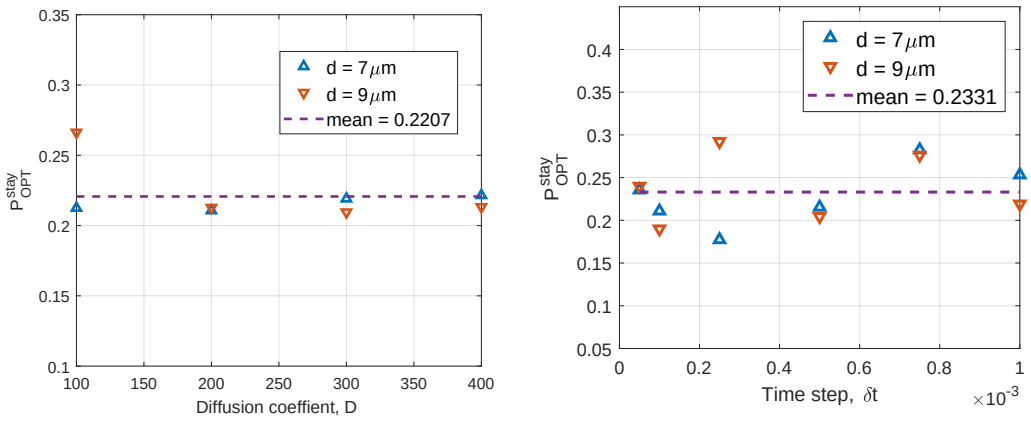


Figure 4.13: $\lambda_{OPT}(\delta t)$ versus the time step and its linear fit in a 3D MCvD system.

4.2.2 Voxel transition probabilities

The probability of staying in the initial voxel for this λ_{OPT} values and each diffusion coefficient, D, and time step, δt, can be calculated, $P^{3D}_{stay,OPT}$, and again the values remain constant within a certain variations due to the resolution of $\lambda_{OPT}(D)$ and $\lambda_{OPT}(\delta t)$. This can be seen in Fig. 4.14.



(a) Varying the diffusion coefficient, D.

(b) Varying the time step, δt.

Figure 4.14: $P^{3D}_{stay,OPT}$ for variations in the environment parameters.

In 2D the values for $P^{2D}_{stay,OPT}$ was similar for both parameters, and the same should be expected in 3D. In this case, the differences between both is higher, but still similar, suggesting again a constant value for $P^{3D}_{stay,OPT}$: It is noticeable that the values for variations in the time step have generally higher variance respect to the mean.

The mean value for any of both simulations, comparing with its equivalent in 2D, which was 0.2990, is smaller. This is related with the extra dimension particles have to diffuse to and, even though the optimum voxel length

is larger, the proportion of particles that leave it is higher.

The general accuracy for the 3D results seem lower than 2D, with greater variance in the results varying the parameters, higher NMSE minimum values and lower higher P-values: the number of released molecules, as mentioned before, seems not to be enough compared with 2D, and the resolution should be higher. For the current project that has not been possible due to the computational time it would consume. Results shown here serve as a first approximation to 3D voxel-based MCvD simulations.

4.3 Flow

Two sets of simulations were done adding the flow: One with a fixed norm for the added flow and changing its direction, and another with a fixed angle but different norms of the flow.

As seen in the previous sections comparing both neighbor models for the voxel-tracking simulations, there was no improvement in the NMSE value for the λ_{OPT} when diagonal neighbors are considered, and, besides, λ_{OPT}^{with}, for a certain set of fixed D, d and δt, has a smaller value than λ_{OPT}^{wout}, which causes its computational time to increase. Therefore, the flow version of the voxel based simulator has only been implemented not considering diagonal neighbors, as the same results can be reached with it but in a significantly smaller time.

To make results for the KS test clearer, the significance level was changed to 0.01.

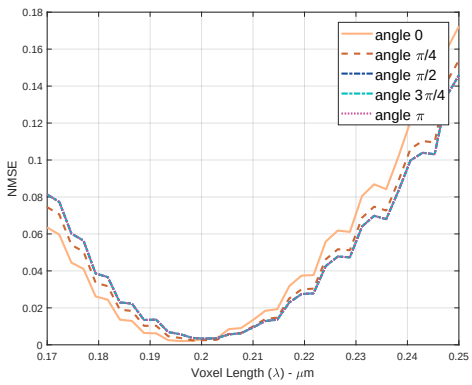
4.3.1 Different flow angles

Five equally spaced flow angles between 0 and π angles where simulated, angle 0 being a flow in the positive x-axis direction, directed to the receiver, and π directed in the opposite way. All the simulation parameters appear at Table 4.13.

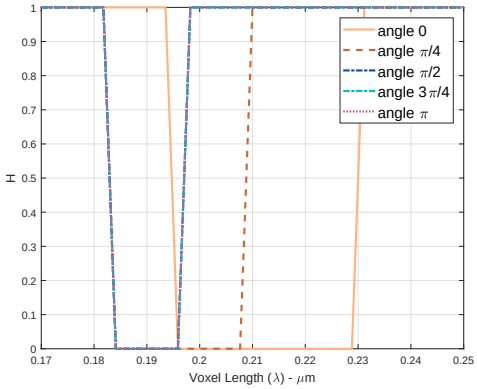
Table 4.13: Simulation parameters varying the angle of the flow

Parameter	Value
Distance (d)	7µm
Diffusion coefficient (D)	100µm^2/s
Simulation time step (δt)	0.0001s
li Number of released molecules (N^{Tx})	100 000
Voxel length (λ^{trial})	$0.17 \sim 0.37$µm, $\Delta\lambda = 0.0105$
Flow angle (norm = 1µm s^{-1})	$\{0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi\}$rad

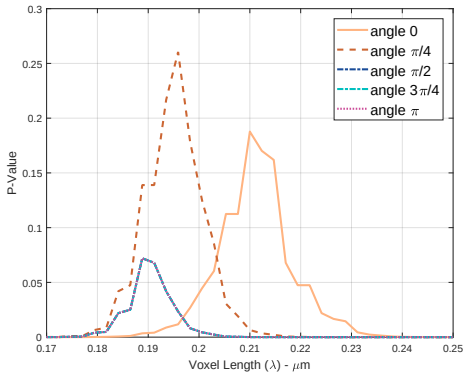
Results for λ_{OPT} from Fig. 4.15 are listed in Table 4.14. Both λ_{OPT}^{NMSE} and λ_{OPT}^{KS} give similar results for each flow angle. It is noticeable that for angles $\frac{\pi}{2}$, $\frac{3\pi}{4}$ and π, for which the flow contribution in the x-axis direction is null or



(a) NMSE plots for different flow angles.



(b) P-value from the KS test.



(c) H-value from the Ks test.

Figure 4.15: Results from the test for MCvD systems with flow at different angles.

negative, the results give the same values in both test for the three simulations, and also that the P-value, even though passing the test, is smaller than the other two angles.

Also, in this case λ_{OPT}^{NMSE} gives very similar results for all angles, while in λ_{OPT}^{KS} there is more variation between values. The region accepted by the KS test H-values follows this trend, with the ones for $\frac{\pi}{2}$, $\frac{3\pi}{4}$ and π even not accepting the λ_{OPT}^{NMSE}.

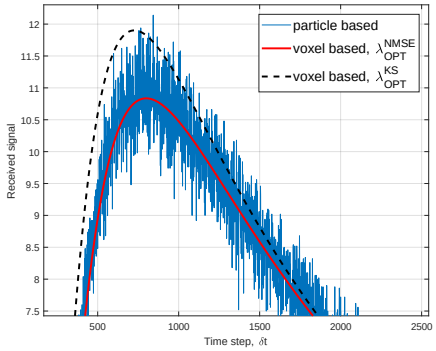
Comparing with the same set of parameters with no flow, Table 4.4, for which λ_{OPT}^{NMSE} and λ_{OPT}^{KS} were 0.2338 and 0.2447, respectively, adding flow lowers the λ_{OPT} in both cases, for this flow norm and the angles analyzed.

Comparing the received signals for the different λ_{OPT} for each flow angle, Fig. 4.16, the one from λ_{OPT}^{NMSE} fits better the peak in all cases, with the received signal from λ_{OPT}^{KS} being as close as both λ_{OPT} values are.

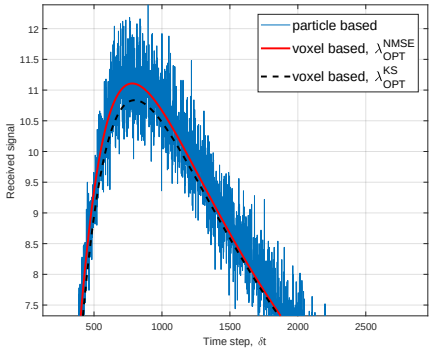
Comparing these signals to those from a simulation with the same of parameters, not considering diagonals, but no flow, in Fig. 4.3a, the signals does not differ a lot from it, as the peak happens in both cases around 0.08s and has a value, in the maximum, between $10 \sim 12$.

Table 4.14: λ_{OPT} from NMSE and KS tests, for varying angle flow.

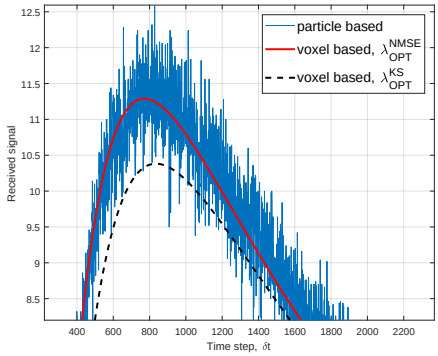
Flow angle (rad)	$\lambda_{OPT}^{NMSE}(\mu m)$	NMSE	$\lambda_{OPT}^{KS}(\mu m)$	P-value
0	**0.1959**	0.002039	**0.2100**	0.1877
$\frac{\pi}{4}$	**0.1982**	0.002422	**0.1959**	0.2605
$\frac{\pi}{2}$	**0.2006**	0.00334	**0.1888**	0.07199
$\frac{3\pi}{4}$	**0.2006**	0.00334	**0.1888**	0.07199
π	**0.2006**	0.00334	**0.1888**	0.07199



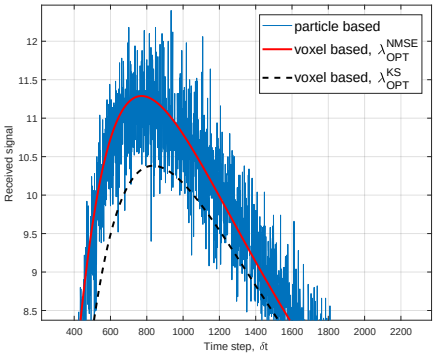
(a) Flow angle 0rad.



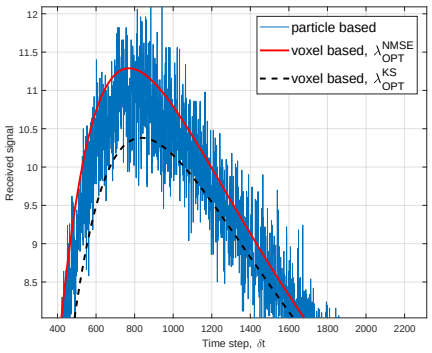
(b) Flow angle $\frac{\pi}{4}$rad.



(c) Flow angle $\frac{\pi}{2}$rad.



(d) Flow angle $\frac{3\pi}{4}$rad.



(e) Flow angle πrad.

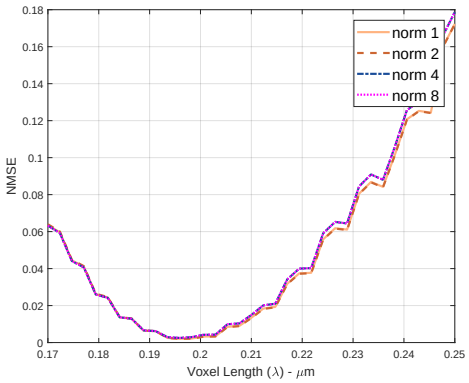
Figure 4.16: Received signals for particle based and voxel based λ_{OPT}^{NMSE} and λ_{OPT}^{KS} for different flow angles.

4.3.2 Different flow norms

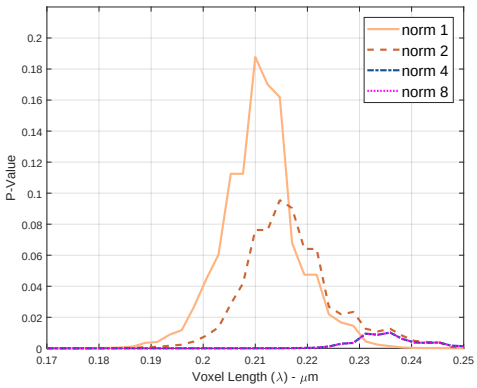
With a fixed angle equal to 0rad, the flow being directed towards the receiver, four different set of simulations, varying the norm of this flow, were made. The parameters are in Table 4.15.

Table 4.15: Simulation parameters varying the norm of the flow.

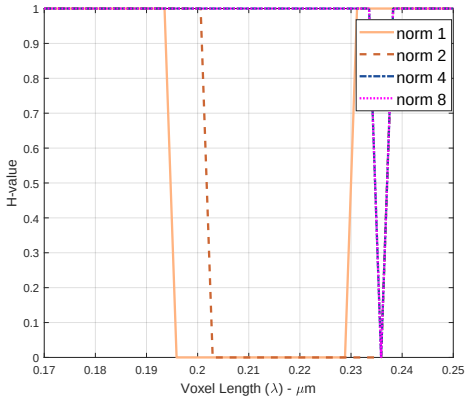
Parameter	Value
Distance (d)	7µm
Diffusion coefficient (D)	$100\mu m^2/s$
Simulation time step (δt)	0.0001s
Number of released molecules (N^{Tx})	100 000
Voxel length (λ^{trial})	$0.17 \sim 0.37\mu m$, $\Delta\lambda = 0.0105$
Flow norm (angle = 0rad)	$\{1, 2, 4, 8\}\mu m\, s^{-1}$



(a) NMSE plots for different flow norms.



(b) P-value from the KS test.



(c) H-value from the Ks test.

Figure 4.17: Results from the test for MCvD systems with flow with different norms.

Result plots can be seen in Fig. 4.17, and the values for λ_{OPT}^{NMSE} and λ_{OPT}^{KS} for each of the set of simulations appear in Table 4.16.

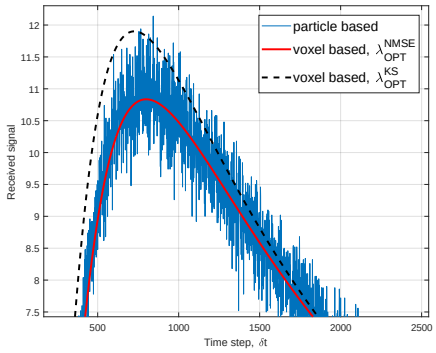
In this case, the NMSE minimum gives the same λ_{OPT}^{NMSE} for the four simulations, although the value of the NMSE varies. The difference with the λ_{OPT}^{KS} are much more notable in this case, with even a difference of 0.04µm for the last two cases. It is important anyway in this case to notice that the P-value for these two simulations is very low, and the KS test H-value, which has only one accepted voxel length, does so due to the change in the significance level, that has been lowered for this analysis.

Table 4.16: λ_{OPT} from NMSE and KS tests, for varying flow norm.

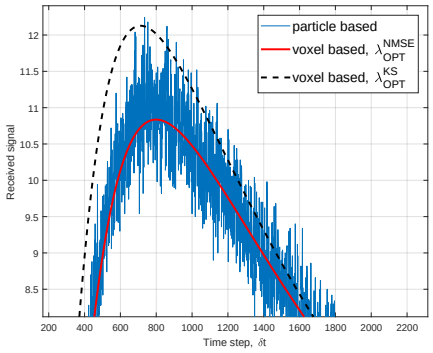
Flow norm (µm s^{-1})	λ_{OPT}^{NMSE}(µm)	NMSE	λ_{OPT}^{KS}(µm)	P-value
1	**0.1959**	0.002039	**0.21**	0.1877
2	**0.1959**	0.001994	**0.2147**	0.09551
4	**0.1959**	0.002502	**0.2359**	0.01013
8	**0.1959**	0.002502	**0.2359**	0.01013

Looking at the λ_{OPT}^{KS}, it grows with the flow norm, even if the P-values get lower, while this is not happening with λ_{OPT}^{NMSE}.

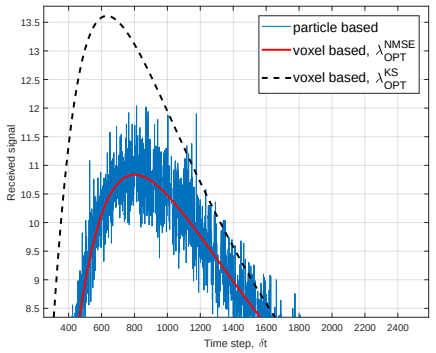
From the accepted region by the H-values for the other two cases, flow norm 1µm s^{-1} λ_{OPT}^{NMSE} is the only one accepted by both test.



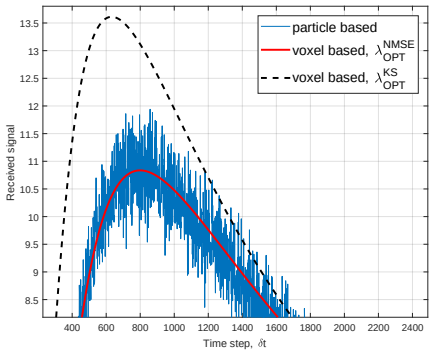
(a) Flow norm 1µm s^{-1}.



(b) Flow norm 2µm s^{-1}.



(c) Flow norm 4µm s^{-1}.



(d) Flow norm 8µm s^{-1}.

Figure 4.18: Received signals for particle based and voxel based λ_{OPT}^{NMSE} and λ_{OPT}^{KS} for different flow angles.

From the figures of the received signals, Fig. 4.18, again minimum NMSE seems to be better at predicting a λ_{OPT} that fits better at the peak. Specially for the last two cases, whose λ_{OPT}^{KS} differed a lot from their λ_{OPT}^{NMSE}, the voxel-based received signals from λ_{OPT}^{KS} fails to give an acceptable fit at the peak, thus not being able to predict the signal.

For the λ_{OPT}^{NMSE}, $P_{stay,OPT}^{2D,flow}$ varying the flow angle gives values between **0.22919** and **0.23734**, while for λ_{OPT}^{KS} this goes between **0.2169** for the lower λ values and **0.25358** for the higher.

Varying the flow norm, $P_{stay,OPT}^{2D,flow}$ the given by λ_{OPT}^{NMSE} has a value of **0.22919**, which is consistent with the previous results varying the parameters, while the λ_{OPT}^{KS} give values of **0.25358**, **0.26166** and **0.29747**, which is much higher.

In both cases λ_{OPT}^{NMSE} being able to fit the peak and having similar $P_{stay,OPT}^{2D,flow}$ values reinforces the idea that this is a constant value, near **0.23**. In this case, the addition of flow, although, as seen in the plots, does not affect significantly the shape of the particle-based signals, it seems affect more the voxel-based, lowering the optimal voxel length, which, for results with no flow, had a value was around **0.299**.

4.4 λ_{OPT} predictions

From the previous sections, we have seen the optimum voxel lengths appears related to a constant value for the probability of stay that only varies with the neighbor model (although it has been seen that the initial model, with no diagonals considered, is enough) and the dimensions of the MCvD environment, $P_{stay,OPT}^{2D}$ and $P_{stay,OPT}^{3D}$.

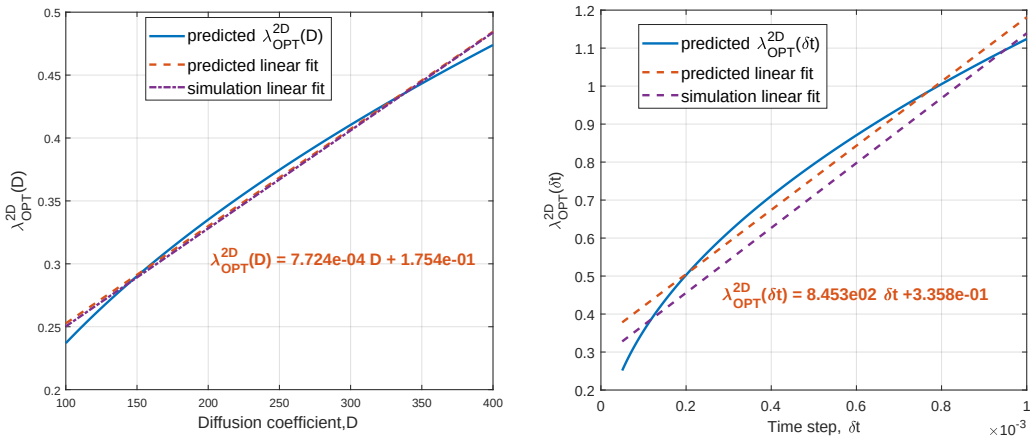
For a certain set of fixed environmental parameters, we could try to predict which will be the λ_{OPT}^{flow} for each of the simulations. We could try to reproduce, using this result, the λ_{OPT} obtained in the previous sections and its relation with the environment parameters.

As the constant value for $P_{stay,OPT}^{2D}$, not considering diagonal neighbors, we can take the mean between this value in the two previous sections, the one from variations in the diffusion coefficient and the one from variations in the time step. This, with the parameter from the table, defines a nonlinear equation as:

$$P_{stay}^{2D}(\lambda_{OPT}) - 0.29918 = 0 \tag{4.1}$$

That can be solved with the function *fsolve()* in Matlab, an has a single defined solution for $\lambda_{OPT}^{predicted}$. Letting only vary the diffusion coefficient or the time step, we get function predicting the λ_{OPT}, which is not completely linear, but we could try to perform a linear fit and compare it to the previous one. This results appear in Fig. 4.19.

It can be clearly seen that the relation between λ_{OPT}^{2D} is not completely linear, specially in the case of δt, which was already hinted in the plots obtained before, Fig. 4.5 and Fig. 4.6. For the same set of parameters, the



(a) Varying the diffusion coefficient, D.

(b) Varying the time step, δt.

Figure 4.19: λ^{2D}_{OPT} predicted for varying D and δt, and their linear fit.

simulation data gave a linear fit with values, for variations in D, $\lambda^{wout}_{OPT}(D) = 7.787e-04D + 1.724e-01$, and in δt, $\lambda^{wout}_{OPT}(\delta t) = 8.538e02\delta t + 2.852e-01$, which are also depicted in the plots: The first one approximates very well the linear fit from the predictions, while the latter shows some displacement. The reason behind this might be that the actual function is far from linear and the simulation δt values weren't equally spaced, as are the ones from these latter results, causing both fits to differ. Nevertheless, the slopes are very similar.

Chapter 5

Conclusions

5.1 Overview of the project

This project's aim was to study and analyze the correctness of a different approach to MC simulations, voxel based simulations instead of particle based, that was hoped to been able to reproduce the same results as the latter, but reducing its runtime for big systems.

Different models for the number of neighbors needed in order to obtain the correct signal were developed and implemented in Matlab, their results for the fitness and runtime analyzed separately and compared. Codes have been implemented for 2D and 3D systems.

Results show the voxel length, λ, is a determinant factor in the fitness of the received signal from the voxel-tracking approach, the one proposed in this project. The optimum value for λ has been analyzed with different tests, and the effect of the environment parameters have on it: Both the diffusion coefficient, D, and the time step, δt, have a similar relation with λ_{OPT}, which for certain window of values can be approximated as linear, while the distance to the receiver, d, as far as this results go, does not affect it.

Both neighbor models and 3D model λ_{OPT} have been shown to reproduce the same behavior under parameter changes, but with different values for λ_{OPT} and therefore in their runtime: It has been seen that considering diagonal neighbors does not lead to any improvement while increases the computational runtime.

It has been discovered that λ_{OPT} is related with a constant value of the probability of stay in the initial voxel for a certain particle in a time step, $P_{OPT,stay}$: This value depends on the neighbor model and the dimensions of the simulator.

This value being a constant allows to, with the environment parameters fixed, find the voxel length that will give the correct received signal while, for systems big enough, reducing the runtime significantly.

The effect of adding flow in a 2D MC environment has also been studied, but further simulations and analysis in that topic may be needed, as the results are not as accurate and conclusive as the previous.

The main results of this project have been submitted to **European View on Molecular Communications, Elsevier Nano Communication Networks**,

under the title **Voxel-Based Simulation Approach for Molecular Communication via Diffusion**, authoring of Dr.H. Birkam Yilmaz, and co-authoring of Dr. Ilker S. Demirkol and Xabier Gutiérrez.

5.2 Future work and perspective

Reducing the runtime of the code may be an objective, since it is something that, besides the idea for the voxel based approach, has not been studied further in this project, and the codes may be improved.

Some of the results of this project, the addition of flow and simulations in a 3D environment, might be analyzed more deeply and with greater accuracy, more complex topologies for the MC environment and bigger systems can be implemented and, with the tools developed during this project, analyze their fitness, the effect they have on λ_{OPT} and how this can be different from the results showed in this thesis.

Appendix A

Simulation software

A.1 Topology generator

A.1.1 board_generate_topology

```
%% Create 2D ENV and Visualize
env_params.D_inMicroMeterSqrPerSecond       = 100;
env_params.voxel_len_inMicroMeter           = 0.5;
env_params.half_voxel_len_inMicroMeter      = env_params.voxel_len_inMi
env_params.env_len_at_each_dir              = 100;

trx_params.tx_center_points                 = [0  0];
trx_params.rx_center_points                 = [7  0];
trx_params.rx_props.type_name               = "square"; % Not used, jus
trx_params.rx_props.type                    = 1;
trx_params.rx_props.side_inMicroMeter       = 4;

sim_params.delta_t                  = 10^-4;
sim_params.num_molecules            = 100;
sim_params.tend_inSeconds           = 2;

% Create VOXEL ENV
voxel_sim_vars = hby_prepare2D_sim_voxels(env_params, trx_params, sim_p
```

A.1.2 hby_prepare2D_sim_voxels

```
function [voxel_sim] = hby_prepare2D_sim_voxels(env_params, trx_params,
% @@ GET INP VARIABLES --START
delta_t                 = sim_params.delta_t;
num_molecules           = sim_params.num_molecules;

D                       = env_params.D_inMicroMeterSqrPerSecond;
voxel_len               = env_params.voxel_len_inMicroMeter;
half_voxel_len          = env_params.half_voxel_len_inMicroMeter;
env_len_at_each_dir     = env_params.env_len_at_each_dir;

tx_center_points        = trx_params.tx_center_points;
```

```
rx_center_points        = trx_params.rx_center_points;
rx_props                = trx_params.rx_props;
% @@ GET INP VARIABLES  --DONE

step_sigma = sqrt(2*D*delta_t);

num_voxel_at_each_dir   = 2 * round( env_len_at_each_dir/voxel_len ) +

num_txs = size(tx_center_points, 1);
% num_rxs = size(rx_center_points, 1); % ToDo: Multi RX LATER

% We add one more voxel to have center at coordinate (0, 0)
center_voxel_idx        = floor( (num_voxel_at_each_dir-1)/2 ) + 1;

% Create 2D Environment
curr_voxel_set_state = zeros(num_voxel_at_each_dir, num_voxel_at_each_d

% Evaluate Duration for Homogenous Voxel (Rough Calculation)
time_stp_shift_for_homogenous_Tx_voxel = round((half_voxel_len/step_sign
time_val_shift_for_homogenous_Tx_voxel = round((half_voxel_len/step_sign

% Prepare TX(s) and curr_voxel_set_state for Voxel-based Sim
tx_voxel_rows = zeros(num_txs,1);
tx_voxel_cols = zeros(num_txs,1);
for ii=1:num_txs
    tx_voxel_idxs = evl_coords_to_2Dvoxel_idxs(tx_center_points(ii,:),
    tx_voxel_rows(ii) = tx_voxel_idxs.row;
    tx_voxel_cols(ii) = tx_voxel_idxs.col;
    curr_voxel_set_state(tx_voxel_rows(ii), tx_voxel_cols(ii)) = num_m
end

% Prepare RX for Voxel-based Sim
% ToDo: Multi RX LATER  (ASSUME Single Rx)
if (rx_props.type == 0)
    fprintf(1, "\n NOT IMPLEMENTED YET");
    %voxel_rx = prepare_CIRCLE_RX( rx_center_points, rx_props, center_v
elseif (rx_props.type == 1)
    voxel_rx = prepare_CUBE_RX( rx_center_points, rx_props, center_voxe
else
    fprintf(1, "\n UnSopported RX Shape Type");
end


% @@ PREPARE/SAVE OUPUT VARIABLES   --START
voxel_sim.env_params        = env_params;
voxel_sim.sim_params        = sim_params;
```

```
voxel_sim.tx_center_points       = tx_center_points;
voxel_sim.rx_center_points       = rx_center_points;
voxel_sim.rx_props               = rx_props;
voxel_sim.voxel_rx               = voxel_rx;

voxel_sim.num_voxel_at_each_dir = num_voxel_at_each_dir;
voxel_sim.center_voxel_idx       = center_voxel_idx;
voxel_sim.curr_voxel_set_state   = curr_voxel_set_state;

voxel_sim.step_sigma             = step_sigma;

voxel_sim.time_stp_shift_for_homogenous_Tx_voxel = time_stp_shift_for_hom
voxel_sim.time_val_shift_for_homogenous_Tx_voxel = time_val_shift_for_hom
% @@ PREPARE/SAVE OUPUT VARIABLES    --DONE
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [voxel_rx] = prepare_CUBE_RX( rx_center_points, rx_props, cent
% (0, 0) is the center of the center voxel
rx_side_len = rx_props.side_inMicroMeter;

rx_center_x_coord = rx_center_points(1);
rx_center_y_coord = rx_center_points(2);

rx_min_x_coord = rx_center_x_coord - rx_side_len/2;
rx_max_x_coord = rx_center_x_coord + rx_side_len/2;
rx_min_y_coord = rx_center_y_coord - rx_side_len/2;
rx_max_y_coord = rx_center_y_coord + rx_side_len/2;

corner_NW_coord = [rx_min_x_coord, rx_max_y_coord]; % TopLeft Corner
corner_SE_coord = [rx_max_x_coord, rx_min_y_coord]; % BottomRight Corne

c_NW_voxel_idxs = evl_coords_to_2Dvoxel_idxs (corner_NW_coord, center_v
c_SE_voxel_idxs = evl_coords_to_2Dvoxel_idxs (corner_SE_coord, center_v

voxel_rx.rows = [];
voxel_rx.cols = [];
voxel_rx.area_ratios = [];
for curr_col = c_NW_voxel_idxs.col:c_SE_voxel_idxs.col
    for curr_row = c_SE_voxel_idxs.row:c_NW_voxel_idxs.row
        voxel_idxs.row = curr_row;
        voxel_idxs.col = curr_col;
        voxel_center_coords = evl_voxel_idxs_to_2Dcenter_coords(voxel_id

        voxel_min_x = voxel_center_coords(1) - voxel_len/2;
        voxel_max_x = voxel_center_coords(1) + voxel_len/2;
```

```
        voxel_min_y = voxel_center_coords(2) - voxel_len/2;
        voxel_max_y = voxel_center_coords(2) + voxel_len/2;

        % Now CONSIDER intersection AREA (https://stackoverflow.com/que
        intsct_area_min_x = max(rx_min_x_coord, voxel_min_x); % left =
        intsct_area_max_x = min(rx_max_x_coord, voxel_max_x); % right =
        intsct_area_min_y = max(rx_min_y_coord, voxel_min_y); % bottom
        intsct_area_max_y = min(rx_max_y_coord, voxel_max_y); % top = m

        intsct_area = (intsct_area_max_x-intsct_area_min_x) * (intsct_ar

        curr_area_ratio = intsct_area / voxel_len^2;

        % Append these values to output
        voxel_rx.rows = [voxel_rx.rows; curr_row];
        voxel_rx.cols = [voxel_rx.cols; curr_col];
        voxel_rx.area_ratios = [voxel_rx.area_ratios; curr_area_ratio];
    end
end

end
```

A.1.3 evl_coords_to_2Dvoxels_idxs

```
function [voxel_idxs] = evl_coords_to_2Dvoxel_idxs (point2Dcoords, cent
% (0, 0) is the center of the center voxel
x_coord = point2Dcoords(1);
y_coord = point2Dcoords(2);

shift_row = round(y_coord/voxel_len);
shift_col = round(x_coord/voxel_len);

voxel_idxs.row = center_voxel_idx + shift_row;
voxel_idxs.col = center_voxel_idx + shift_col;
end
```

A.1.4 evl_voxel_idx_to_2Dcenter_coords

```
function [center_coords2D] = evl_voxel_idxs_to_2Dcenter_coords(voxel_id
% (0, 0) is the center of the center voxel
idx_diff_row = voxel_idxs.row - center_voxel_idx;
idx_diff_col = voxel_idxs.col - center_voxel_idx;

center_x = idx_diff_col * voxel_len;
center_y = idx_diff_row * voxel_len;
```

```
center_coords2D = [center_x , center_y ];
end
```

A.2 Simulator

A.2.1 hby simulate2D particles

```
function [hit_timeline] = hby_simulate2D_particles(env_params, trx_paran
fprintf(1,"\nRUNNING Particle Based Simulator");

if trx_params.rx_props.type == 1
    fprintf(1,"\n*** Rx ShapeType = SQUARE");
    hit_timeline = sim2d_particle_SquareRX_point_src_RUNNER (env_params
elseif trx_params.rx_props.type == 0
    fprintf(1, "\nNOT IMPLEMENTED YET: RX ShapeType = CIRCLE");
else
    fprintf(1, "\nUNSOPPORTED RX Shape Type");
end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [hit_timeline] = sim2d_particle_SquareRX_point_src_RUNNER(env_p
% @@ GET INP VARIABLES --START
D                          = env_params.D_inMicroMeterSqrPerSecond;

tx_center_points           = trx_params.tx_center_points;
rx_center_points           = trx_params.rx_center_points;
rx_props                   = trx_params.rx_props; % FOR SQUARE
rx_side_len                = rx_props.side_inMicroMeter;

delta_t                    = sim_params.delta_t;
tend_inSeconds             = sim_params.tend_inSeconds;
num_molecules              = sim_params.num_molecules;
% @@ GET INP VARIABLES --DONE

% Note: HERE 2D Assumption
rx_min_X = rx_center_points(1) - rx_side_len/2;
rx_max_X = rx_center_points(1) + rx_side_len/2;
rx_min_Y = rx_center_points(2) - rx_side_len/2;
rx_max_Y = rx_center_points(2) + rx_side_len/2;


% Find the number of simulation steps
sim_step_cnt = round(tend_inSeconds / delta_t);
half_sim_step_cnt = round(sim_step_cnt/2);
```

```
% Records the number of molecules at RECEIVER at each time step
hit_timeline = zeros (1, sim_step_cnt);

% Standard deviation of step size of movement N(0,sigma)
sigma = (2*D*delta_t)^0.5;

% Each molecule starts at Tx Loc (default is 0,0 in 2D)
mol_position1 = repmat(tx_center_points, num_molecules, 1);

fprintf(1,"\nStep:%d/%d", 1,sim_step_cnt);
for t=1:sim_step_cnt
    if (t == half_sim_step_cnt )
        fprintf(1,"\nStep:%d/%d", t,sim_step_cnt);
    end
    % propagate the molecules via diffusion
    mol_displace = normrnd (0, sigma, size(mol_position1,1), 2); %2D
    mol_position2 =  mol_position1 + mol_displace;

    % CHECK RECEPTION
    inside_RX_mask = eval_inside_RX_mask(mol_position2, rx_min_X, rx_ma

    % reception (hit) count
    hit_timeline(t) = hit_timeline(t) + nnz(inside_RX_mask);

    %keep the ones indicated by the outside membrane mask
    mol_position2 = mol_position2(~inside_RX_mask, :);

    mol_position1 = mol_position2;
end % END<for t=1:sim_step_cnt>
fprintf(1,"\nStep:%d/%d\n", sim_step_cnt,sim_step_cnt);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [inside_RX_mask] = eval_inside_RX_mask(mol_position2, rx_min_X
x_min_mask = mol_position2(:,2) > rx_min_X;
x_max_mask = mol_position2(:,2) < rx_max_X;

y_min_mask = mol_position2(:,1) > rx_min_Y;
y_max_mask = mol_position2(:,1) < rx_max_Y;

inside_RX_mask = x_min_mask & x_max_mask & y_min_mask & y_max_mask;
end
```

A.2.2 hby simulate2D voxels

```
function [hit_timeline, snapshot_voxel_states] = hby_simulate2D_voxels(
```

```
% @@ GET INP VARIABLES --START
env_params                = voxel_sim_vars.env_params;
sim_params                = voxel_sim_vars.sim_params;

delta_t                   = sim_params.delta_t;
tend_inSeconds            = sim_params.tend_inSeconds;

D                         = env_params.D_inMicroMeterSqrPerSecond;
voxel_len                 = env_params.voxel_len_inMicroMeter;

curr_voxel_set_state      = voxel_sim_vars.curr_voxel_set_state;
voxel_rx                  = voxel_sim_vars.voxel_rx;

time_shift_4homogeneity = voxel_sim_vars.time_stp_shift_for_homogenous_T
% @@ GET INP VARIABLES --DONE

if (numel(snapshot_times) == 0)
    snapshot_times = [0.1 0.2 0.4 0.8 1] * tend_inSeconds;
end

voxel_reside_prob         = evl_pstay2D(voxel_len, D, delta_t);
voxel_leave_prob          = 1 - voxel_reside_prob;

rx_indxs = sub2ind(size(curr_voxel_set_state), voxel_rx.rows, voxel_rx.

% Find the number of simulation steps
sim_step_cnt = round(tend_inSeconds / delta_t);
half_sim_step_cnt = round(sim_step_cnt/2);

% Snapshot time instances
snapshot_sim_step_list = round(snapshot_times / delta_t);

num_snapshots = numel(snapshot_times);
curr_snapshot = 0;

% Records the molecule distribution at snapshot time instances
snapshot_voxel_states = zeros([size(curr_voxel_set_state), num_snapshot

% Records the number of molecules at RECEIVER at each time step
hit_timeline = zeros (1, sim_step_cnt);

fprintf(1,"\nRUNNING Vortex Based Simulator");
fprintf(1,"\nStep:%d/%d", 1,sim_step_cnt);
for t=time_shift_4homogeneity:sim_step_cnt  % shifted START for Homoger
    if (t == half_sim_step_cnt )
        fprintf(1,"\nStep:%d/%d", t,sim_step_cnt);
    end
```

```
    % Evaluate EXITING NumMolecules
    voxel_set_exit_molecules = curr_voxel_set_state * voxel_leave_prob;

    % Remove Exiting Molecules
    voxel_set_after_exit = curr_voxel_set_state - voxel_set_exit_molecu

    % Evaluate ENTERING NumMolecules
    voxel_set_entering_mask2sum = eval_entering_mask2sum(voxel_set_exit_

    % Current State AFTER MOVING/EXITING+ENTERING
    curr_voxel_set_state = voxel_set_after_exit + voxel_set_entering_mas

    % CALCULATE RX Molecules
    rx_voxel_set_state_before_absorption = curr_voxel_set_state(rx_indx
    rx_voxel_absorbed = rx_voxel_set_state_before_absorption .* voxel_r
    % Update current state
    curr_voxel_set_state(rx_indxs) = rx_voxel_set_state_before_absorptio
    % Record the number of NRX molecules
    nrx_curr_t = sum(rx_voxel_absorbed);
    hit_timeline(t) = nrx_curr_t;

    if ismember(t, snapshot_sim_step_list)
        % take snapshot of this state
        curr_snapshot = curr_snapshot +1;
        snapshot_voxel_states(:,:,curr_snapshot) = curr_voxel_set_state
    end
end % END<for t=1:sim_step_cnt>
fprintf(1,"\nStep:%d/%d\n", sim_step_cnt, sim_step_cnt);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [voxel_exit_mask2sum] = eval_entering_mask2sum(voxel_exit_mole
[rr, cc] = size(voxel_exit_molecules);

shifted_right = [zeros(rr,1) voxel_exit_molecules(:,1:end-1)/4];
shifted_left = [voxel_exit_molecules(:,2:end)/4 zeros(rr,1)];

shifted_up = [voxel_exit_molecules(2:end,:)/4; zeros(1,cc)];
shifted_down = [zeros(1,cc); voxel_exit_molecules(1:end-1,:)/4];

voxel_exit_mask2sum = shifted_right + shifted_left + shifted_up + shifte
end
```

A.2.3 hby simulate2D voxels diagonal

```
function [hit_timeline, snapshot_voxel_states] = hby_simulate2D_voxels_
```

```
% @@ GET INP VARIABLES --START
env_params                = voxel_sim_vars.env_params;
sim_params                = voxel_sim_vars.sim_params;

delta_t                   = sim_params.delta_t;
tend_inSeconds            = sim_params.tend_inSeconds;

D                         = env_params.D_inMicroMeterSqrPerSecond;
voxel_len                 = env_params.voxel_len_inMicroMeter;

curr_voxel_set_state      = voxel_sim_vars.curr_voxel_set_state;
voxel_rx                  = voxel_sim_vars.voxel_rx;

time_shift_4homogeneity = voxel_sim_vars.time_stp_shift_for_homogenous_
% @@ GET INP VARIABLES --DONE

if (numel(snapshot_times) == 0)
    snapshot_times = [0.1 0.2 0.4 0.8 1] * tend_inSeconds;
end

voxel_reside_prob         = evl_pstay2D(voxel_len, D, delta_t);
voxel_leave_prob          = 1 - voxel_reside_prob;

[p_aligned,p_diagonal]    = evl_pstay2D_diag(voxel_len, D, delta_t);
prop                      = p_aligned/p_diagonal;
voxel_aligned_prob        = prop/(prop + 1);
voxel_diagonal_prob       = 1/(prop + 1);

rx_indxs = sub2ind(size(curr_voxel_set_state), voxel_rx.rows, voxel_rx.

% Find the number of simulation steps
sim_step_cnt = round(tend_inSeconds / delta_t);
half_sim_step_cnt = round(sim_step_cnt/2);

% Snapshot time instances
snapshot_sim_step_list = round(snapshot_times / delta_t);

num_snapshots = numel(snapshot_times);
curr_snapshot = 0;

% Records the molecule distribution at snapshot time instances
snapshot_voxel_states = zeros([size(curr_voxel_set_state), num_snapshot

% Records the number of molecules at RECEIVER at each time step
hit_timeline = zeros (1, sim_step_cnt);

fprintf(1,"\nRUNNING Vortex Based Simulator");
```

```
fprintf(1,"\nStep:%d/%d", 1,sim_step_cnt);
for t=time_shift_4homogeneity:sim_step_cnt  % shifted START for Homoger
    if (t == half_sim_step_cnt )
        fprintf(1,"\nStep:%d/%d", t,sim_step_cnt);
    end
    % Evaluate EXITING NumMolecules
    voxel_set_exit_molecules = curr_voxel_set_state * voxel_leave_prob;

    % Remove Exiting Molecules
    voxel_set_after_exit = curr_voxel_set_state - voxel_set_exit_molecu

    % Evaluate ENTERING NumMolecules
    voxel_set_entering_mask2sum = eval_entering_mask2sum(voxel_set_exit_

    voxel_set_entering_mask2sum_diagonal = eval_entering_mask2sum_diago

    % Current State AFTER MOVING/EXITING+ENTERING
    curr_voxel_set_state = voxel_set_after_exit + voxel_set_entering_mas

    % CALCULATE RX Molecules
    rx_voxel_set_state_before_absorption = curr_voxel_set_state(rx_indx
    rx_voxel_absorbed = rx_voxel_set_state_before_absorption .* voxel_r
    % Update current state
    curr_voxel_set_state(rx_indxs) = rx_voxel_set_state_before_absorptio
    % Record the number of NRX molecules
    nrx_curr_t = sum(rx_voxel_absorbed);
    hit_timeline(t) = nrx_curr_t;

    if ismember(t, snapshot_sim_step_list)
        % take snapshot of this state
        curr_snapshot = curr_snapshot +1;
        snapshot_voxel_states(:,:,curr_snapshot) = curr_voxel_set_state
    end
end % END<for t=1:sim_step_cnt>
fprintf(1,"\nStep:%d/%d\n", sim_step_cnt, sim_step_cnt);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [voxel_exit_mask2sum] = eval_entering_mask2sum(voxel_exit_mole
[rr, cc] = size(voxel_exit_molecules);

shifted_right = [zeros(rr,1) voxel_exit_molecules(:,1:end-1)/4];
shifted_left = [voxel_exit_molecules(:,2:end)/4 zeros(rr,1)];

shifted_up = [voxel_exit_molecules(2:end,:)/4; zeros(1,cc)];
shifted_down = [zeros(1,cc); voxel_exit_molecules(1:end-1,:)/4];
```

```
voxel_exit_mask2sum = voxel_aligned_prob *(shifted_right + shifted_left
end

function [voxel_exit_mask2sum_diagonal] = eval_entering_mask2sum_diagon
[rr, cc] = size(voxel_exit_molecules);

shifted_right = [zeros(rr,1) voxel_exit_molecules(:,1:end-1)/4];
shifted_left = [voxel_exit_molecules(:,2:end)/4 zeros(rr,1)];

shifted_right_up = [shifted_right(2:end,:); zeros(1,cc)];
shifted_right_down = [zeros(1,cc);shifted_right(1:end-1,:)];

shifted_left_up = [shifted_left(2:end,:); zeros(1,cc)];
shifted_left_down = [zeros(1,cc);shifted_left(1:end-1,:)];

voxel_exit_mask2sum_diagonal = voxel_diagonal_prob*(shifted_right_up +
+ shifted_left_down);
end
```

A.2.4 hby_simulate2D_particles_flow

```
function [hit_timeline] = hby_simulate2D_particles_flow(env_params, trx_
fprintf(1,"\nRUNNING Particle Based Simulator");

if trx_params.rx_props.type == 1
    fprintf(1,"\n*** Rx ShapeType = SQUARE");
    hit_timeline = sim2d_particle_SquareRX_point_src_RUNNER (env_params
elseif trx_params.rx_props.type == 0
    fprintf(1, "\nNOT IMPLEMENTED YET: RX ShapeType = CIRCLE");
else
    fprintf(1, "\nUNSOPPORTED RX Shape Type");
end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [hit_timeline] = sim2d_particle_SquareRX_point_src_RUNNER(env_
% @@ GET INP VARIABLES  --START
D                          = env_params.D_inMicroMeterSqrPerSecond;

tx_center_points           = trx_params.tx_center_points;
rx_center_points           = trx_params.rx_center_points;
rx_props                   = trx_params.rx_props; % FOR SQUARE
rx_side_len                = rx_props.side_inMicroMeter;
```

```
delta_t                 = sim_params.delta_t;
tend_inSeconds          = sim_params.tend_inSeconds;
num_molecules           = sim_params.num_molecules;
% @@ GET INP VARIABLES  --DONE

% Note: HERE 2D Assumption
rx_min_X = rx_center_points(1) - rx_side_len/2;
rx_max_X = rx_center_points(1) + rx_side_len/2;
rx_min_Y = rx_center_points(2) - rx_side_len/2;
rx_max_Y = rx_center_points(2) + rx_side_len/2;


% Find the number of simulation steps
sim_step_cnt = round(tend_inSeconds / delta_t);
half_sim_step_cnt = round(sim_step_cnt/2);

% Records the number of molecules at RECEIVER at each time step
hit_timeline = zeros (1, sim_step_cnt);

% Standard deviation of step size of movement N(0,sigma)
sigma = (2*D*delta_t)^0.5;

% Each molecule starts at Tx Loc (default is 0,0 in 2D)
mol_position1 = repmat(tx_center_points, num_molecules, 1);

% Flow
flow_displace = v_flow*delta_t;

fprintf(1,"\nStep:%d/%d", 1,sim_step_cnt);
for t=1:sim_step_cnt
    if (t == half_sim_step_cnt )
        fprintf(1,"\nStep:%d/%d", t,sim_step_cnt);
    end
    % propagate the molecules via diffusion
    mol_displace = normrnd (0, sigma, size(mol_position1,1), 2); %2D

    mol_position2 =  mol_position1 + mol_displace + flow_displace;

    % CHECK RECEPTION
    inside_RX_mask = eval_inside_RX_mask(mol_position2, rx_min_X, rx_ma

    % reception (hit) count
    hit_timeline(t) = hit_timeline(t) + nnz(inside_RX_mask);

    %keep the ones indicated by the outside membrane mask
    mol_position2 = mol_position2(~inside_RX_mask, :);
```

```
    mol_position1 = mol_position2;
end % END<for t=1:sim_step_cnt>
fprintf(1,"\nStep:%d/%d\n", sim_step_cnt,sim_step_cnt);
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [inside_RX_mask] = eval_inside_RX_mask(mol_position2, rx_min_X
x_min_mask = mol_position2(:,2) > rx_min_X;
x_max_mask = mol_position2(:,2) < rx_max_X;

y_min_mask = mol_position2(:,1) > rx_min_Y;
y_max_mask = mol_position2(:,1) < rx_max_Y;

inside_RX_mask = x_min_mask & x_max_mask & y_min_mask & y_max_mask;
end
```

A.2.5 hby_simulate2D_voxels_flow

```
function [hit_timeline, snapshot_voxel_states] = hby_simulate2D_voxels_
% @@ GET INP VARIABLES --START
env_params                = voxel_sim_vars.env_params;
sim_params                = voxel_sim_vars.sim_params;

delta_t                   = sim_params.delta_t;
tend_inSeconds            = sim_params.tend_inSeconds;

D                         = env_params.D_inMicroMeterSqrPerSecond;
voxel_len                 = env_params.voxel_len_inMicroMeter;

curr_voxel_set_state      = voxel_sim_vars.curr_voxel_set_state;
voxel_rx                  = voxel_sim_vars.voxel_rx;

time_shift_4homogeneity = voxel_sim_vars.time_stp_shift_for_homogenous_
% @@ GET INP VARIABLES --DONE

if (numel(snapshot_times) == 0)
    snapshot_times = [0.1 0.2 0.4 0.8 1] * tend_inSeconds;
end

[p_stay, p_right, p_left, p_up, p_down]        = eval_pstay2D_flow(voxel_
voxel_leave_prob          = 1 - p_stay;

sum_p = p_right + p_left + p_up + p_down;

alpha_stay = p_stay/sum_p;
alpha_right = p_right/sum_p;
alpha_left = p_left/sum_p;
```

```
alpha_up = p_up/sum_p;
alpha_down = p_down/sum_p;

rx_indxs = sub2ind(size(curr_voxel_set_state), voxel_rx.rows, voxel_rx.

% Find the number of simulation steps
sim_step_cnt = round(tend_inSeconds / delta_t);
half_sim_step_cnt = round(sim_step_cnt/2);

% Snapshot time instances
snapshot_sim_step_list = round(snapshot_times / delta_t);

num_snapshots = numel(snapshot_times);
curr_snapshot = 0;

% Records the molecule distribution at snapshot time instances
snapshot_voxel_states = zeros([size(curr_voxel_set_state), num_snapshot

% Records the number of molecules at RECEIVER at each time step
hit_timeline = zeros (1, sim_step_cnt);

fprintf(1,"\nRUNNING Vortex Based Simulator");
fprintf(1,"\nStep:%d/%d", 1,sim_step_cnt);
for t=time_shift_4homogeneity:sim_step_cnt  % shifted START for Homoge
    if (t == half_sim_step_cnt )
        fprintf(1,"\nStep:%d/%d", t,sim_step_cnt);
    end
    % Evaluate EXITING NumMolecules
    voxel_set_exit_molecules = curr_voxel_set_state * voxel_leave_prob;

    % Remove Exiting Molecules
    voxel_set_after_exit = curr_voxel_set_state - voxel_set_exit_molecu

    % Evaluate ENTERING NumMolecules
    voxel_set_entering_mask2sum = eval_entering_mask2sum(voxel_set_exit_

    % Current State AFTER MOVING/EXITING+ENTERING
    curr_voxel_set_state = voxel_set_after_exit + voxel_set_entering_mas

    % CALCULATE RX Molecules
    rx_voxel_set_state_before_absorption = curr_voxel_set_state(rx_indx
    rx_voxel_absorbed = rx_voxel_set_state_before_absorption .* voxel_r
    % Update current state
    curr_voxel_set_state(rx_indxs) = rx_voxel_set_state_before_absorptio
    % Record the number of NRX molecules
    nrx_curr_t = sum(rx_voxel_absorbed);
    hit_timeline(t) = nrx_curr_t;
```

```
    if ismember(t, snapshot_sim_step_list)
        % take snapshot of this state
        curr_snapshot = curr_snapshot +1;
        snapshot_voxel_states(:,:,curr_snapshot) = curr_voxel_set_state
    end
end % END<for t=1:sim_step_cnt>
fprintf(1,"\nStep:%d/%d\n", sim_step_cnt, sim_step_cnt);

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [voxel_exit_mask2sum] = eval_entering_mask2sum(voxel_exit_mole
[rr, cc] = size(voxel_exit_molecules);

shifted_right = alpha_right * [zeros(rr,1) voxel_exit_molecules(:,1:end
shifted_left = alpha_left * [voxel_exit_molecules(:,2:end) zeros(rr,1)]

shifted_up = alpha_up * [voxel_exit_molecules(2:end,:); zeros(1,cc)];
shifted_down = alpha_down * [zeros(1,cc); voxel_exit_molecules(1:end-1

voxel_exit_mask2sum = shifted_right + shifted_left + shifted_up + shifte
end
```

A.2.6 evl_pstay2D

```
function [p_stay] = evl_pstay2D(L_voxel, D, delta_t)

sigma = sqrt(2*D*delta_t);
denumerator = sigma * sqrt(2);

fun = @(x,y) (1/4) * ( erf((L_voxel-x)/denumerator)-erf(-x/denumerator)

p_stay = integral2(fun, 0,L_voxel, 0,L_voxel) / L_voxel^2;
end
```

A.2.7 evl pstay2D diag

```
function [p_aligned,p_diagonal] = evl_pstay2D_diag(L_voxel, D, delta_t)

sigma = sqrt(2*D*delta_t);
denumerator = sigma * sqrt(2);

fun_aligned = @(x,y) (1/4) * ( erf((2*L_voxel-x)/denumerator)-erf((L_vo

fun_diagonal = @(x,y) (1/4) * ( erf((2*L_voxel-x)/denumerator)-erf((L_v
```

```
p_aligned = integral2(fun_aligned, 0,L_voxel, 0,L_voxel) / L_voxel^2;

p_diagonal = integral2(fun_diagonal, 0,L_voxel, 0,L_voxel) / L_voxel^2;
end
```

A.2.8 eval_pstay2D_flow

```
function [p_stay, p_right, p_left, p_up, p_down] = eval_pstay2D_flow(L_

v_x = v_flow(1);
v_y = v_flow(2);

sigma = sqrt(2*D*delta_t);
denumerator = sigma * sqrt(2);

fun_stay = @(x,y) (v_x * delta_t + (1/2) * ( erf((L_voxel-x)/denumerato

fun_right = @(x,y) (v_x * delta_t + (1/2) * ( erf((2*L_voxel-x)/denume

fun_left = @(x,y) (v_x * delta_t + (1/2) * ( erf(-x/denumerator)-erf((-

fun_up = @(x,y) (v_x * delta_t + (1/2) * ( erf((L_voxel-x)/denumerator)-

fun_down = @(x,y) (v_x * delta_t + (1/2) * ( erf((L_voxel-x)/denumerato

p_stay = integral2(fun_stay, 0,L_voxel, 0,L_voxel) / L_voxel^2;

p_right = integral2(fun_right, 0,L_voxel, 0,L_voxel) / L_voxel^2;
p_left = integral2(fun_left, 0,L_voxel, 0,L_voxel) / L_voxel^2;
p_up = integral2(fun_up, 0,L_voxel, 0,L_voxel) / L_voxel^2;
p_down = integral2(fun_down, 0,L_voxel, 0,L_voxel) / L_voxel^2;
end
```

List of Figures

List of Tables

Bibliography

[1] William C Agosta. *Chemical communication: the language of pheromones*. Henry Holt and Company, 1992.

[2] Ian F Akyildiz, Fernando Brunetti, and Cristina Blázquez. "Nanonetworks: A new communication paradigm". In: *Computer Networks* 52.12 (2008), pp. 2260–2279.

[3] A Bruce et al. "Molecular Biology of the Cell 5th edn (New York: Garland Science)". In: (2007).

[4] Yansha Deng et al. "Modeling and simulation of molecular communication systems with a reversible adsorption receiver". In: *IEEE Transactions on Molecular, Biological and Multi-Scale Communications* 1.4 (2015), pp. 347–362.

[5] Nariman Farsad et al. "A comprehensive survey of recent advancements in molecular communication". In: *IEEE Communications Surveys & Tutorials* 18.3 (2016), pp. 1887–1919.

[6] Weisi Guo et al. "Molecular versus electromagnetic wave propagation loss in macro-scale environments". In: *IEEE Transactions on Molecular, Biological and Multi-Scale Communications* 1.1 (2015), pp. 18–25.

[7] Satoshi Hiyama and Yuki Moritani. "Molecular communication: Harnessing biochemical materials to engineer biomimetic communication systems". In: *Nano Communication Networks* 1.1 (2010), pp. 20–30.

[8] Changmin Lee et al. "Machine learning based channel modeling for molecular MIMO communications". In: *arXiv preprint arXiv:1704.00870* (2017).

[9] Milica Stojanovic. "Acoustic (underwater) communications". In: *Encyclopedia of Telecommunications* (2003).

[10] H Birkan Yilmaz and Chan-Byoung Chae. "Simulation study of molecular communication systems with an absorbing receiver: Modulation and ISI mitigation techniques". In: *Simulation Modelling Practice and Theory* 49 (2014), pp. 136–150.