HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

Jan Peter Drees

A Computationally Inexpensive and Flexible Simulation Model for Molecular Communication

Master's Thesis in Computer Science

23 December 2019

Please cite as:
Jan Peter Drees, "A Computationally Inexpensive and Flexible Simulation Model for Molecular Communication," Master's Thesis (Masterarbeit), Heinz Nixdorf Institute, Paderborn University, Germany, December 2019.

Distributed Embedded Systems (CCS Labs)
Heinz Nixdorf Institute, Paderborn University, Germany

Fürstenallee 11 · 33102 Paderborn · Germany

http://www.ccs-labs.org/

A Computationally Inexpensive and Flexible Simulation Model for Molecular Communication

Master's Thesis in Computer Science

vorgelegt von

Jan Peter Drees

geb. am [redacted]
in [redacted]

angefertigt in der Fachgruppe

Distributed Embedded Systems
(CCS Labs)

Heinz Nixdorf Institut
Universität Paderborn

Betreuer:	**Florian Klingler**
Gutachter:	**Falko Dressler**
	Holger Karl
Abgabe der Arbeit:	**23. Dezember 2019**

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.
Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties.
I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Jan Peter Drees)
Paderborn, 26 February 2020

Abstract

In some environments wireless communication with electromagnetic radiation is unfeasible or simply too dangerous. Molecular communication (MolCom) works by the sender emitting particles instead and is therefore a promising alternative. A carrier medium, such as a gas or liquid, physically moves the particles through space. The presence of particles is then sensed at receivers, which recover the transmitted data. In an industrial macro-scale MolCom system, existing pipe systems could be used, into which magnetic particles are injected. Current simulators for this either lack the capability to consider the complex flow inside the pipe system or suffer from poor performance. I present the "Pogona" simulator which uses computational fluid dynamics (CFD) to model the flow of liquid inside the pipe system. The flow is imported into the simulator and used to predict the movement of particles. Additionally, models for the injection and sensing of particles are proposed. To improve performance and flexibility, a scene approach that allows the configuration of scenarios consisting of independent objects is proposed. I verified my simulator prototype with simple scenarios that are compared to analytical results and known-good implementations. Simulation scenarios that match an existing testbed were created and their output compared to empirical measurements. Overall, a low degree of agreement was achieved and room for improvement remains. My analysis shows that the interpolation algorithm applied to the CFD flow output is of importance. However, finding well-suited interpolation algorithms remains an open research task.

Kurzfassung

Klassische Drahtloskommunikation mithilfe von elektromagnetischer Strahlung ist in einigen Umgebungen kaum möglich oder zu riskant. Molekulare Kommunikation, bei der stattdessen Partikel vom Sender ausgestoßen werden, stellt eine interessante Alternative dar. Die Partikel werden vom Träger-Medium, z.B. einem Gas oder einer Flüssigkeit, durch den Raum bewegt. Der Empfänger detektiert das Vorhandensein der Partikel und dekodiert die damit übertragene Nachricht. Im industriellen Kontext können für diese Art der Kommunikation magnetische Partikel in bestehende Rohrsysteme eingespritzt werden. Aktuelle Simulatoren sind entweder nicht in der Lage, die komplexen Strömungen innerhalb der Rohre berücksichtigen, oder sie erfordern einen hohen Rechenaufwand. Der von mir präsentierte “Pogona”-Simulator verwendet einen Simulator für numerische Strömungsmechanik (engl. CFD), um diese Strömung zu berechnen. Diese Strömung wird importiert und benutzt, um effizient die Bewegung der Partikel vorherzusagen. Außerdem werden Modelle für die Einspritzung und das Detektieren der Partikel vorgestellt. Um die Flexibilität und Performance zu erhöhen, wurde ein Szenen-System mit voneinander unabhängigen Objekten entwickelt. Ich habe den Simulator verifiziert, indem ich diesen mit analytischen Ergebnissen oder bestehenden Simulatoren in vereinfachten Szenarios verglichen habe. Zudem wurde ein bestehendes Experiment modelliert und die Messungen mit dem Simulationsergebnis verglichen. Dabei konnte nur eine mittelmäßige Übereinstimmung festgestellt werden und weitere Verbesserungen bleiben nötig. Ein großer Einfluss des Interpolationsalgorithmus auf das Ergebnis wurde identifiziert. Passende Algorithmen für diese Anwendung werden benötigt und stellen den Gegenstand zukünftiger Forschung dar.

Contents

Chapter 1

Introduction

Wireless communication is usually implemented utilizing electromagnetic radiation at radio frequencies. The sender and receiver employ antennas, with the receiver sensing the changes in the electromagnetic field introduced by the sender. Molecular communication works with an entirely different paradigm based on the transport of particles. Here, the sender disperses particles into the medium, e.g. air, and the receiver detects the concentration with a sensor. This means that the particles move physically from the sender to the receiver, which is caused by diffusion, the flow of the medium, or propulsion systems.

In some scenarios, traditional radio communication is just not viable. The propagation of radio waves is severely reduced by the environment inside air vents, underground mines, or oil pipelines. In other cases, for example in explosive gases, the electric circuitry required is prohibited. Currently, molecular communication is not used in such industrial scenarios, but appears promising to solve these issues.

Simulation could help in investigating the usefulness of such systems for networking on a bigger scale. Particle transport in fluid flows needs to be considered in the underlying model, since it massively influences the channel characteristics.

The existing simulators [1]–[6] have already been used for molecular communication (MolCom) on nano-scales. Particle motion caused by diffusion and simplified fluid flows is captured by their models already. Bronner and Dressler [7] conclude that they currently lack the capability to consider particle motion caused by more complex fluid flows.

Well-established tools for predicting complex fluid flow are computational fluid dynamics (CFD) simulations. They are able to predict the fluid movement, pressure, temperature, and other characteristics by solving fundamental equations of fluid dynamics numerically. The continuous distribution of fluid characteristics is mapped onto a grid, enabling a solver to work on discrete grid nodes only. Single-phase CFD models are appropriate for scenarios where a single fluid is present, in our case

where a tube is filled entirely with water. Multi-phase CFD models are appropriate for scenarios where different fluids or particles interact, for example the sedimentation of sand in a river. These are more difficult to model accurately and even more computationally demanding than single-phase simulations [8]. In theory, such models would be appropriate to model the particles being carried along with the water in the tube. Unfortunately, they were never intended to be used for interactive simulations. Thus, choosing to trigger a new transmission based on received data is not possible. Additionally, even transmitting a single bit of information with these models would require hours of CPU time to simulate.

Another model proposed specifically for macro-scale MolCom by Wicke et al. [9] abstracts the pipe flow as an ideal parabolic flow profile and derives an analytic solution from it. This approach is very efficient, but only covers ideal, simplified systems, such as straight pipes. Furthermore, the injection pattern of the particles is not modeled and requires manual fitting of experimental data.

Instead, I investigate the approach of exporting the resulting fluid motion from a single-phase CFD simulation and using this as the input for a new simulator. This approach has recently been proposed by Bronner and Dressler [7]. Inside the simulator, the flow of the surrounding medium is used to predict the movement of discrete particles injected for communication purposes. This core is accompanied by models for particle injection and sensing. The combined model aims towards a compromise between accuracy, flexibility, and computational complexity. It is supposed to be less computation-heavy and more easily integratable than a multi-phase CFD model. However, this will necessitate some sacrifices regarding accuracy. On the other hand, it will be more computation-heavy than analytic solutions. The aim is to support complex geometries and also predict the injection, thereby improving the accuracy compared to simple models. This gives the model a wider range of applicability compared to the restricted set of ideal geometries possible in analytic models.

The remainder of this thesis is organized as follows. First, some fundamentals are covered in Chapter 2. This includes related work on MolCom, fluid dynamics basics, CFD, and the algorithms necessary to predict particle movement. Chapter 3 then presents the simulator prototype that was implemented, as well as the underlying models. It starts with a high-level overview and goes into details of channel, sender and receiver model where necessary. In Chapter 4 this simulator is then evaluated. It focuses on correctness of implementation, followed by performance analysis and comparison with empirical results. Chapter 5 then concludes the thesis by generalizing the evaluation results and pointing towards future work.

Chapter 2

Fundamentals

In the following Section 2.1, the basic concept of MolCom is explained. It also gives an overview of the current progress of the field of MolCom at macro-scales, including simulation approaches. In particular, the testbed presented by Unterweger et al. [10] is explained, as the model is compared against measurements from this setup. Because it consists of circular tube sections, Section 2.2 covers some fundamentals of fluid dynamics for this case. Another important tool for determining fluid dynamics in arbitrary geometries is CFD. It is covered in detail in Section 2.3. Because the simulator works in discrete time steps, the prediction of molecule movement depends on discrete integration algorithms, which are presented in Section 2.4. Interpolation, which Section 2.5 covers, is also necessary since the output of the CFD simulation is spatially discrete. Finally, the quantitative analysis of the channel impulse response (CIR) as predicted by the simulator is needed. Metrics for this are given in Section 2.6.

2.1 Molecular Communication

MolCom is an active interdisciplinary research topic. The underlying concept is the physical movement of perturbations in a medium from the sender to the receiver. This broad paradigm can be implemented on a range of scales and with different technologies.

For example, hormones are used for communication inside the human body [11]. These are carried by the blood stream to the receptors inside organs, where they control physiological processes. This inspired the use of MolCom for nano-scale networks, for example consisting of nanobots inside living tissue.

Only recently, such systems were proposed to be used on a larger scale. A first experiment was conducted by Farsad, Guo, and Eckford [12], who injected small droplets of ethanol into the air of a room. A breathalizer-like sensor, which could detect the changes in ethanol concentration, was placed on the other side of the

room. To aid the movement of the droplets towards the sensor, a ventilator was used. Encoding digital data is then done by dividing time in discrete slots of a few seconds length. Each slot then contains the transmission of a single bit. A bit with value “1” is transmitted by injecting during a time slot, while no injection occurs for a bit with value “0”. The receiver can then measure the concentration during each injection time slot to recover the original bit stream. If the concentration increases, this is interpreted as a “1” bit, while a decrease is interpreted as a “0” bit. The authors demonstrated this by transmitting a short text message across the otherwise empty room.

Jamali et al. [13] have surveyed macro-scale testbeds and found several different approaches. Some of these are biological, for example using *E. coli* bacteria or cells. Others are synthetic, for example using flourescent dye in a water chamber or smoke vortex rings in open air.

The system that will be considered in this thesis is communication inside an industrial pipe system containing a flowing liquid. This has been investigated experimentally by Unterweger et al. [10], utilizing particles injected into flowing water at the sender and detecting the magnetic susceptibility inside of the pipe at the receiver. Their experimental setup is shown in Figure 2.1. They use peristaltic pumps for the continuous background flow and the injection. Their sensor is the commercial MS2G single frequency susceptibility sensor produced by Bartington. It has a sensor rate of around 10 Hz. The particles are custom-made “LA38” superparamagnetic iron oxide nanoparticles (SPIONs) with lactic acid coating. Superparamagnetism

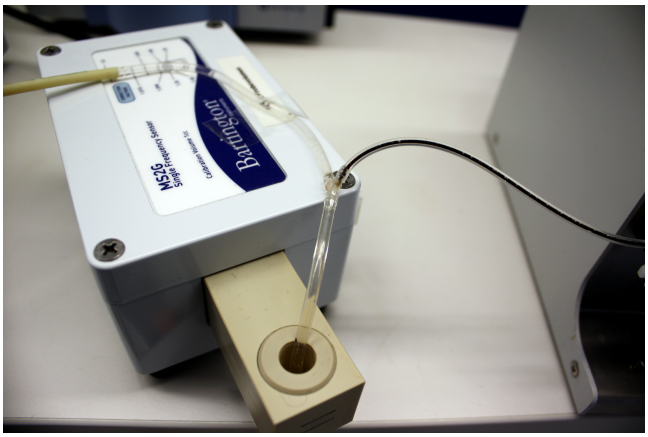


Figure 2.1 – Close-up view of the channel, transmitter and receiver of the Erlangen experimental setup. The Y-piece in the tube is the injection site of the transmitter. The pump on the right regulates the injection, pumping the black nanoparticles to the Y-piece, where they enter the background stream of water passing in the transparent tube. The grey box contains the sensor electronics of the receiver, the measurements are done where the tube passes through the sand-colored casing.

means that the contained iron will exhibit behavior comparable to a magnet when an external magnetic field is applied, but remain non-magnetic otherwise. This results in both a large signal spike when the particles pass through the susceptometer and low clustering of particles.

In the following, the term “molecule” will be used when referring to what the IEEE standard 1906.1-2015 [14] defines as a “Message Carrier”. What exactly a molecule represents will depend on the specific application scenario. It could be a single atom, a grain of sediment, a nano-machine, a droplet of fluerescent fluid, or a bacterium. When modeling the Erlangen testbed, SPIONs are the molecules.

2.1.1 Simulation

Several simulators for MolCom have been put forth. Most of these are specifically aimed at simulating nano-scale systems. In such systems, the influence of diffusion is dominant, while the influence of flow becomes of a higher importance on larger scales.

Several approaches for simulation are possible. A microscopic simulation tracks the movement of individual molecules over discrete time steps. A mesoscopic simulator only considers a more coarse-grained level of detail. For example, one approach is to split the domain into cells, with only the aggregated concentration of molecules in each cell being used. Another option is to forego this molecule tracking entirely, by using an analytical description of the channel behavior instead.

Bronner and Dressler [7] have conducted a survey of existing simulators. Their analysis concludes that NanoNS3, BiNS2, and AcCoRD all offer diffusion and flow modeling. However, they are lacking in the possible flow complexity. AcCoRD is only able to consider a constant global flow vector. BloodVoyagerS is specialized for the human cardiovascular system and uses a constant flow vector that depends on the diameter of the blood vessel. BiNS2 also supports flow inside a circular pipe segment, which it uses for blood vessel simulations. NanoNS3 is different in this regard, as it uses an analytical description of the propagation characteristics. This means that it could possibly support more complex flows, as long as the corresponding analytical description were known.

Commercial multiphysics simulators like COMSOL are also used for analysis. These promise a high level of accuracy, but suffer from long run-times necessary to obtain the results.

Wicke et al. [9] developed a model for the Erlangen testbed that utilizes transfer functions. They were able to derive these from a theoretic model of fluid flow in a circular pipe. However, such approaches are difficult to apply to more complex geometries where an analytic solution for the flow does not exist. Furthermore, they had to manually fit the injection pattern of the particles to their experimental data.

Fabian Bronner proposed an alternative approach using pre-calculated vector fields to track molecule movement in a simulator [7]. His approach is what this thesis builds upon, utilizing the output of a dedicated fluid simulator.

2.2 Fluid Dynamics in Circular Pipes

The basic fluid dynamic in a circular pipe is dominated by the friction of the fluid at the pipe walls. Shear forces cause a velocity of zero right at the pipe wall, while the fluid in the pipe center displays the highest flow speed. Near the pipe wall, where the viscosity has the largest effect, a boundary layer forms.

The following formulas are analytically derived for the case of a single fluid moving continuously in a circular pipe. Additionally, the length of the pipe needs to be sufficiently larger than the radius, and the disturbances in the area of an in- or outlet are not considered.

The flow profile is parabolic when investigating a cut-through section. Bird, Stewart, and Lightfoot [15, §5.1] give Equation 2.1 for determining it. R is the total pipe radius, while r is the distance of the location to the pipe center. The parabola is scaled with the maximum velocity $v_{z,max}$.

$$v_z = v_{z,max}\left[1 - \left(\frac{r}{R}\right)^2\right] \tag{2.1}$$

They also give Equation 2.2 for the relation between $v_{z,max}$ and the average velocity $\langle v_z \rangle$.

$$\langle v_z \rangle = \frac{1}{2} v_{z,max} \tag{2.2}$$

2.3 Computational Fluid Dynamics

Determining the flow of fluids given some external constraints is a long-standing task in disciplines like mechanical engineering. Analytical solutions exist for some geometries, such as the flow inside a circular pipe presented in Section 2.2. With the advent of modern computers, it has become feasible to find approximate numerical solutions for more complex geometries where analytical solutions are near-impossible to obtain. This area of research is called CFD. In the following, the material of Ferziger and Perić [16] is used when describing how a CFD simulation works.

The foundations for each CFD simulation are differential equations describing the system behavior. In theory, these would be the full set of Navier-Stokes equations used to describe the motion of viscous fluids accurately. For practical applications, approximations enable the use of a reduced set of the Navier-Stokes equations or alternative models. The domain is split into individual regions or “cells” and time is

discretized into time steps. This discretization in time and space allows a reduction of the partial differential equations to algebraic equations. For each time step and cell, a numerical solution for the equations is determined.

Of paramount importance is the provision of so-called boundary conditions. For example, the inlet of a pipe might have a user-specified flow rate. Another important boundary condition is the no-slip condition. It states that the fluid infinitesimally close to a wall is motionless because of friction. A CFD simulator will then try to find a flow inside the pipe that satisfies these conditions while also satisfying the equations. Over several time steps of a laminar flow simulation, the solution that the simulator comes up with converges towards an unchanging state. Ideally, the equations fully hold for the resulting flow at the end of the simulation. An ideal match is usually not possible, so the aim is to reduce the residual error in the balance of the equations.

An important component of the CFD simulator is the solver. It specifies in which order and how the equations are to be solved within a time step. These steps are usually repeated until either the residual error is smaller than a specified threshold or a cutoff number of iterations has been reached. The solver algorithm does not affect the converged result of the CFD simulation, but it does influence the speed of convergence.

2.3.1 Turbulence

When fluid layers flow along without disturbances, the resulting flow is called laminar. Fluid flow where the flow pattern is complex and time-dependent instead is called turbulent.

Turbulence is a very complex phenomenon and still an open problem in physics. The Reynolds number is employed to get an approximation of whether a flow is laminar. It quantifies the ratio between the inertial and viscous forces in the fluid. Bird, Stewart, and Lightfoot [15, p. 52] give Equation 2.3 to calculate it in the case of a circular tube. $\langle v_z \rangle$ is the average flow velocity in tube direction, ρ is the density, and μ is the dynamic viscosity of the fluid, while D denotes the internal tube diameter.

$$\mathrm{Re} = \frac{D\langle v_z \rangle \rho}{\mu} \tag{2.3}$$

According to Bird, Stewart, and Lightfoot [15], a flow with *Re* of less than about 2100 is laminar, which enables the usage of simpler flow analysis methods compared to turbulent flows.

In our scenario, there are particles suspended in the fluid when an injection is happening. Agrawal, Choueiri, and Hof [17] determined experimentally that the

presence of suspended particles also influences turbulence. When varying the concentration of the suspended particles, they found concentration-dependent changes in turbulence. However, their results indicate that at a *Re* of less than 800, the differences are negligible even for concentrations as high as 25 %.

For turbulent flow, the variations across a wider range of temporal and spatial scales necessitate alternative approaches for CFD. Solving the full Navier-Stokes equations in the way outlined in Section 2.3 is called direct numerical simulation (DNS). The space and time resolution required for DNS is orders of magnitude larger than the resolution necessary for laminar flow. This usually makes the computation and memory requirements of the simulation prohibitive. Alternative approaches use turbulence models to reduce the required resolution and improve the practicality of turbulent CFD simulation. However, these do not achieve the same level of accuracy as DNS.

2.3.2 Mesh Generation

Since the fluid simulation only works on discrete cells, the subdivision of the target geometry into several cells has to be determined. The quality of the fluid simulation heavily depends on the quality of the underlying mesh. The term mesh resolution is used to describe the subdivision of the entire object into mesh cells. If the cells are smaller in relation to the object, then the mesh resolution is higher. A mesh with higher resolution will produce more fine-grained results, at the cost of higher

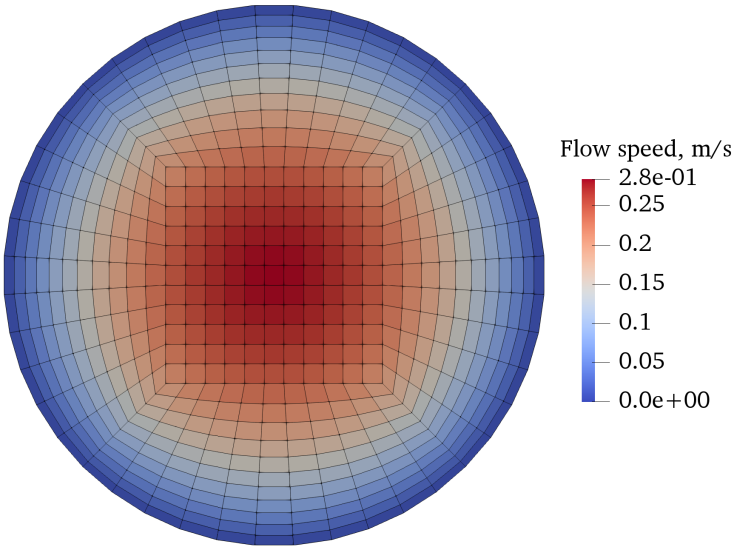


Figure 2.2 – Cross section view of a manually-defined O-ring tube mesh. Cell coloring is based on flow speed.

computation duration and memory footprint. However, the smaller cells also require a finer time resolution, further increasing the duration.

Figure 2.2 shows what a manually created tube mesh with an O-Ring structure around a grid core looks like. This structure is being recommended for the use in pipes[1]. It features refined cells towards the pipe wall, where the boundary layer needs to be resolved. Near the center, where the gradient between neighboring cells is low, such a fine resolution is not required.

For general shapes without symmetry or easily defined blocks, the manual approach quickly becomes very labor-intensive, so automated tools are preferable. Automated mesh generation is possible with the OpenFOAM tool snappyHexMesh[2]. With this utility, a regular tetrahedal mesh is generated, and then intersected with the shape from a 3D object. This shape can be generated using an arbitrary 3D modeling software. The resulting mesh is then automatically refined at the surface and finally "snapped" to it. In Figure 2.3 the result of snappyHexMesh for a tube section is shown. It can be observed that most of the object consists of simple regular grid cells, while the snapping algorithm takes care of the boundaries.

2.3.3 Nearest Neighbor Search

Vector fields are the mathematical concept underlying CFD data structures. For discrete positions in n-dimensional space, vectors are defined. In our case, for 3-

[1] `http://www.cfdyna.com/Home/OpenFOAM.html`
[2] `https://cfd.direct/openfoam/user-guide/v7-snappyhexmesh`

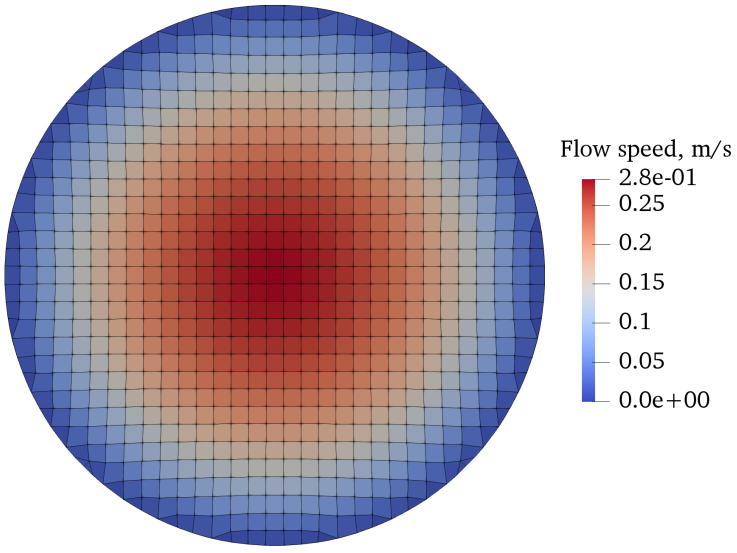


Figure 2.3 – Cross section view of a tube mesh generated with snappyHexMesh. Cell coloring is based on flow speed.

dimensional spatial coordinates of each mesh cell, a 3-dimensional vector is defined as the flow (movement) of the fluid in this cell. This is the structure in which most CFD simulators export their output. Alternatively, only a scalar value is given for each position. For example, the pressure at a position does not possess any directionality and is thus represented in this format.

When operating in a vector field, it is necessary to determine the discrete vector field point closest to an arbitrary query point. This problem is called "Nearest Neighbor Search". The general problem may be restated as to determine the m nearest neighbor positions to the query point in k dimensions.

A naive solution is to use a linear search. This works by determining the distance from the query point to every vector field point. Then, the m positions with the smallest distance are returned. This algorithm has linear run-time.

Space partitioning data structures represent point data by slicing k-dimensional space into regions. This way, points that are close together spatially are in the same or adjacent regions. A query algorithm then only needs to determine the regions close to the query point and can disregard all others. Within a region, traditional algorithms like linear search are then performed to determine an exact solution.

Bentley [18] proposed the "k-d tree", which is a special case of such a space partitioning data structure. It slices each region in half by selecting a splitting plane through a point. All points in the "left" half are stored in the left subtree of the tree node. All points in the "right" half are stored in the right sub-tree. This is applied to the sub-trees recursively, until each point is stored in a node. The splitting plane normal vectors are chosen alternatingly between the k axes. This simplifies tree operations, as simple coordinate comparisons can be used to determine the region of a point.

According to Friedman, Bentley, and Finkel [19], creation of a k-d tree has a run-time of $kn \log n$ for n points. Searching for the m closest points has an expected run-time with relationship $\log n$ to the overall number of data points n.

2.4 Integration of Molecule Movement

Taking the vector field output of the CFD model and applying it to particle motion in discrete time steps also creates errors. These errors are inherent to discrete steps and well known for numerical integration. The simplest method by Euler takes the flow at the current particle position p_n and follows this direction until the next time step [20]. Equation 2.4 is used for this, predicting the next position p_{n+1} from the

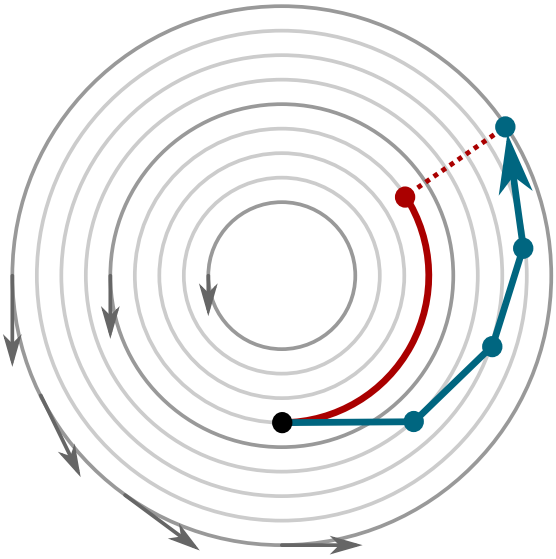


Figure 2.4 – Integration of a particle (black) moving in a rotating field (grey circles) using the Euler method (path shown in blue). The integrated position differs from the true position (red), resulting in a substantial integration error (dashed line).

value of the function f at the current position. h denotes the step size equal to $t_{n+1} - t_n$.

$$p_{n+1} = p_n + hf(t_n, p_n) \tag{2.4}$$

Figure 2.4 illustrates one problem of the Euler method. There, the circular movement underlying the vector field is not replicated by the particle, which follows a diverging spiral instead. The error for Euler's method can be made arbitrarily small by using more fine-grained time steps h, but that increases the computational effort by the same factor.

The Runge-Kutta integration method has a rich history, going back to the original 1895 publication of Runge [21]. Butcher [22] present the different methods of this family that aim to decrease the error in Eulers method. They build onto the Euler method, but iteratively retrieve the flow at the predicted new position and intermediate steps and consider all for the overall integration. Equations 2.5 through 2.8 show this iterative lookup for four iterations where k_1 is equal to the original Euler step. Equation 2.9 is then used to weight the intermediate values and predict the next position.

$$k_1 = h\, f(t_n, p_n) \tag{2.5}$$

$$k_2 = h\, f\left(t_n + \frac{h}{2}, p_n + \frac{k_1}{2}\right) \tag{2.6}$$

$$k_3 = h\, f\left(t_n + \frac{h}{2}, p_n + \frac{k_2}{2}\right) \tag{2.7}$$

$$k_4 = h\, f\,(t_n + h, p_n + k_3) \tag{2.8}$$

$$p_{n+1} = p_n + \tfrac{1}{6}k_1 + \tfrac{1}{3}k_2 + \tfrac{1}{3}k_3 + \tfrac{1}{6}k_4 \tag{2.9}$$

2.5 Flow Interpolation

The output of the fluid simulator is the discrete flow in entire mesh cells, which only approximates the continuous distribution in the real world. A molecule is usually not in the center of a mesh cell, but moves through it at different positions. It is therefore necessary to interpolate the flow at sampling positions where the fluid simulation makes no explicit prediction.

The simplest approach would be to assume the flow is constant within each cell. This means we can simply use the flow associated to the cell that the sampling position is inside of. However, the output of this algorithm will poorly match any flow containing gradients, producing discrete jumps at cell borders instead.

One advanced algorithm that aims to eliminate these discontinuities while maintaining the underlying flow structure is to apply inverse distance weighting to all flow values. The approach by Shepard [23] works on scatted data points and takes the distance to the known data points into account when determining the interpolated value at the sample location. Its definition is given in Equation 2.10, where x is the lookup point, $x_1, \ldots, x_n$ are the known points, while $f(x_i)$ are the associated function values. The normalized weights given in Equation 2.11 are used, which ensures that they sum up to 1. The weights are simply the inverse distances as given in Equation 2.12, using an exponent μ and calculated with the Euclidean norm $|\cdot|$.

$$S_\mu[f](x) = \sum_{i=1}^{n} \widetilde{W}_{\mu,i}(x) f(x_i) \tag{2.10}$$

$$\widetilde{W}_{\mu,i}(x) = \frac{W_{\mu,i}(x)}{\sum_{k=1}^{n} W_{\mu,k}(x)} \tag{2.11}$$

$$W_{\mu,i}(x) = |x - x_i|^{-\mu} \tag{2.12}$$

In the original version, an exponent of $\mu = 2$ was chosen. This worked well for the 2D map interpolation Shepard was working on [23]. However, the general form allows for arbitrary exponents, as long as they satisfy $\mu > 0$.

This approach considers all known data points, which can be a performance issue for large data sets. Additionally, a large exponent needs to be picked. The reason for this was pointed out by Han-Kwang Nienhuys on the Wikipedia page on inverse distance weighting (IDW)[3]. He shows that the exponent should be bigger than the spatial dimension, because otherwise the summed weights of far-away values will be bigger than the summed weights of the nearest neighbors. A modified version of this proof using 3 dimensions is given in Appendix A. It shows that an exponent below 3 is not suitable for large data sets.

An alternative that only needs to consider local information is to define an area of influence around the sampling point and only using the known data points inside of it. This raises the questions of determining the area of influence and weighing the elements inside of it. A simple approach would be to define a fixed sphere of influence and applying Shepard's method inside of it. Another would be to define a desired number of elements inside the sphere and dynamically scaling the sphere depending on the local neighborhood. Smaller exponents can be used in this approach, since far-away values have no influence.

Franke and Nielson [24] have proposed using Equation 2.13 as a replacement for the original inverse distance weighting. It also weights the distances with exponent μ, but substracts the radius R_{w_i} of the sphere of influence first. This ensures that the resulting interpolant remains smooth. If the original weights are used, discontinuities appear when new values enter or leave the sphere of influence. With the modified weights, these are initially not considered. For them, the distance $|x - x_i|$ equals the influence radius R_{w_i}, resulting in a weight of $W_{\mu,i}(x) = 0$.

$$W_{\mu,i}(x) = \left(\frac{1}{|x - x_i|} - \frac{1}{R_{w_i}} \right)^{\mu} \tag{2.13}$$

The choice for R_{w_i} is a trade-off. The bigger it is chosen, the smoother the interpolant becomes. This also results in increased computation cost and underprediction of extreme values. Chosen too small, the interpolant is not as smooth or no neighbors might be found at all.

To address differing sample densities or scales, N_w, the number of nearest neighbors to consider, is defined instead. R_{w_i} is then chosen in such a way that the sphere of influence includes the desired number of neighbors. This can be done by determining the N_W nearest neighbors and setting R_{w_i} to the largest distance of any of them. One aspect to consider is that due to the weighting function reaching 0 at distance R_{w_i}, the most distant point is not actually incorporated into the interpolation. Only

[3] `https://en.wikipedia.org/wiki/Inverse_distance_weighting#Basic_form`

at most $N_w - 1$ neighbors will be taken into account, which means that $N_w = 9$ considers the eight nearest neighbors.

In theory, only cells that are adjacent to the lookup point in the underlying CFD mesh structure would need to be considered for flow interpolation. In a regular grid, there would be eight of these cells and trilinear interpolation could be used. However, IDW can be used for arbitrary irregular meshes as well. Wilhelms et al. [25] successfully used eight nearest neighbor IDW for interpolation in hexahedral meshes. They picked eight since these are the the cube vertices for regular grids. However, they point out that for arbitrary hexahedral cells, the nearest neighbors might not match the cube vertices.

2.6 Channel Impulse Response Metrics

Several metrics exist for assessing the CIR. To illustrate this, Figure 2.5 shows a CIR example with the various metrics drawn in.

In general, τ specifies the delay between the very first received signal component and some other component. These multipath components are usually discrete, for example caused by the reflection of radio waves. For MolCom, the delays until the particles are sensed at the receiver can be interpreted as components.

The delay of the maximum peak component relative to the first received signal component is defined as the τ_{max}.

The simplest is the maximum amplitude, which is shown as V_{max}. Another important metric for inter-symbol interference (ISI) is the delay spread, shown as τ in the figure. Goldsmith [26, p. 58] defines the delay spread as “the time delay

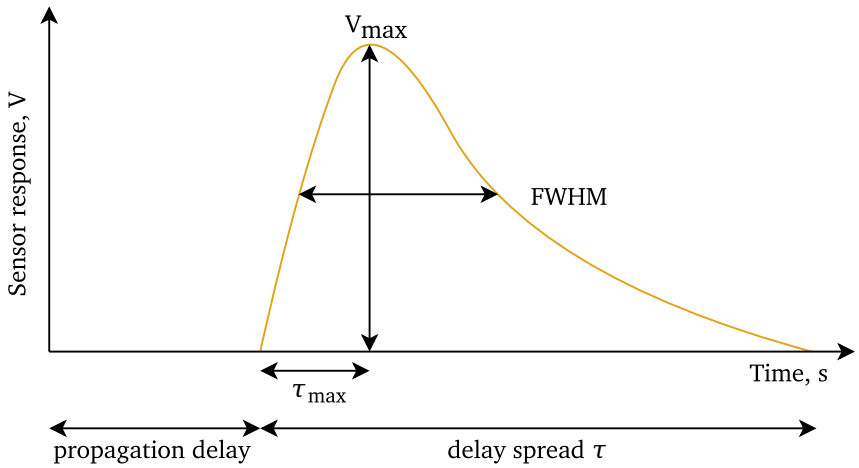


Figure 2.5 – Example channel impulse response.

between the arrival of the first received signal component [...] and the last received signal component associated with a single transmitted pulse." In the case of particles in a pipe, this is difficult to determine, as the delay spread is, in theory, infinite. This is caused by particles close to the pipe wall that have an arbitrarily small movement speed, resulting in an arbitrarily large time of arrival at the receiver. For practical reasons, we therefore only consider the delay spread of portions of the signal which reach above the noise floor to analyze the simulations.

Another metric is the full width at half maximum (FWHM). For each CIR peak, the time delay between the first time the received signal raises above 50 % of the V_{max} and the last time it stays above it is determined.

A metric that summarizes the delay behavior is the root mean squared (RMS) delay spread. It quantifies the deviation from the average delay. Equation 2.14 gives the definition used by Goldsmith [26, p. 87] for the RMS delay spread. It measures the deviation from the average delay spread μ_{T_m} of the power delay profile $A_c(\tau)$.

$$\sigma_{T_m} = \sqrt{\frac{\int_0^\infty \left(\tau - \mu_{T_m}\right)^2 A_c(\tau) d\tau}{\int_0^\infty A_c(\tau) d\tau}} \tag{2.14}$$

Chapter 3

The Pogona Simulator

The core idea for the MolCom model consists of predicting particle movement based on the flow output of a CFD simulation. The full simulator then consists of CFD, sender, receiver, and channel models. Such a simulator architecture, allowing integration of different models, has been created and is described in Section 3.1. This architecture along with selected models were implemented in a Python 3 prototype. I collaborated with Fabian Bronner and Lukas Stratmann in implementing the simulator we decided to name "Pogona". The core functionality providing the supporting framework for the models is described in Section 3.2. Section 3.3 explains how the real-world particles are represented in the simulation. The remainder of this chapter presents the actual models and their Pogona implementations. It starts with the channel model for movement prediction in Section 3.4, followed by the transmitter modeling in Section 3.5. This chapter concludes with the different receiver models for particle sensing in Section 3.6.

3.1 Architecture

The core simulation components and their interaction during a simulation time step are displayed in the diagram in Figure 3.1. How they work together and what their respective responsibilities are will be covered step by step in this chapter.

Initially, the simulator reads the configuration files containing the model setup and respective parameters. The simulator components are then created and initialized. This initialization procedure encompasses reading the CFD result files and setting up the necessary data structures.

At a high-level view, a time step then consists of the following. First, the transmitters create new molecules. Then, the movement of all molecules during the time step is calculated by the channel components. If a molecule is positioned in a receiver's detection area at the end of the time step, the receiver considers the

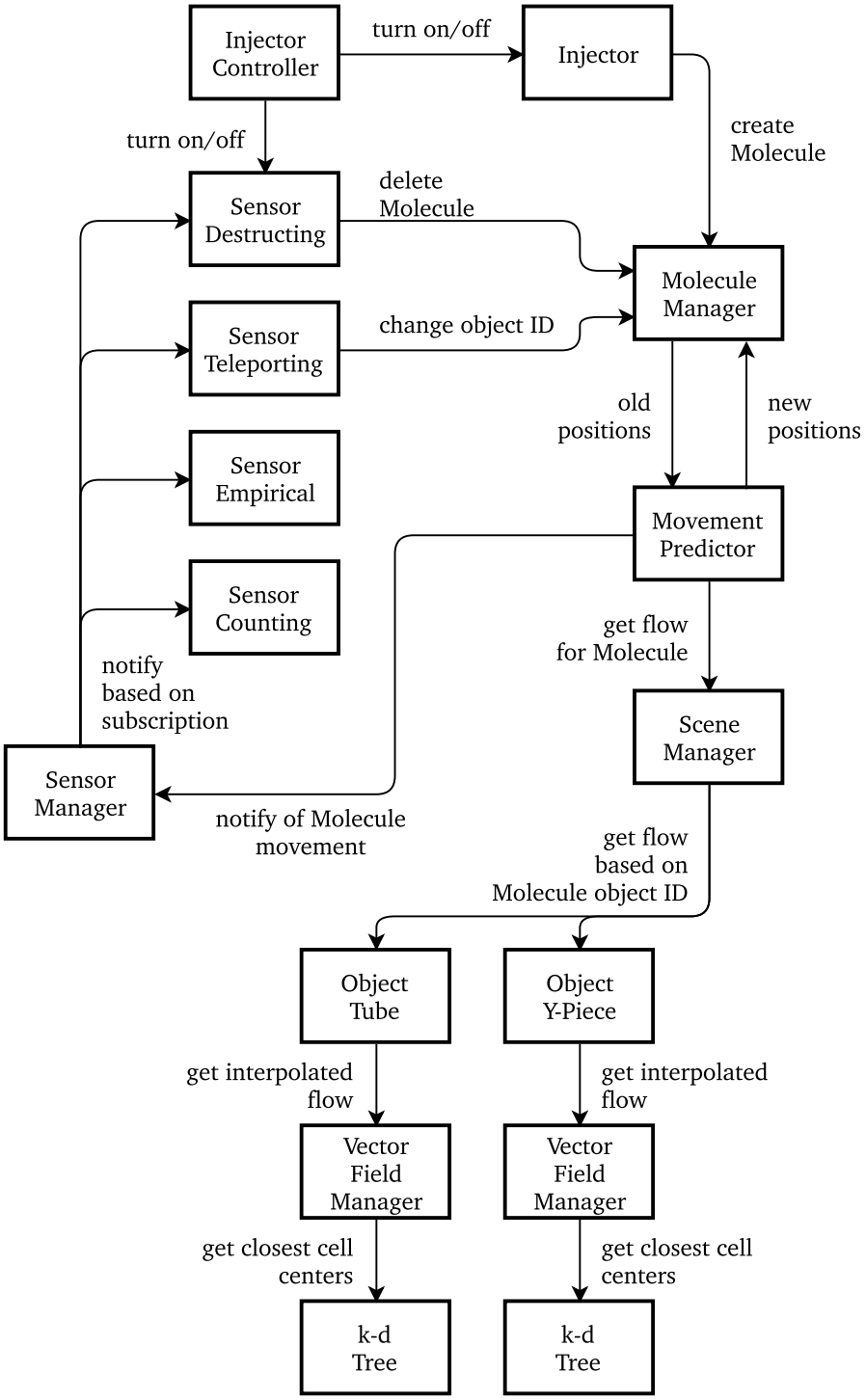


Figure 3.1 – Diagram of component interactions within a time step.

molecule for the reported value. Finally, the simulation time is incremented by the duration of the time step by the simulation kernel. This loop is repeated until the specified simulation duration is reached. The non-aggregated component output can be written to file during this simulation loop. This is done for the current position of each molecule, facilitating post-processing with 3D visualization software.

At the end of the simulation, every component is notified. This is used to write the aggregated sensor observations from the simulation to a file, which can later be analyzed with other tools.

3.2 Framework

To facilitate flexibility in the implementation of the architecture described in Section 3.1, a supporting framework is needed. This means providing a class with base functionality for the various simulation components that a new component can inherit from. This `Component` approach is presented in Section 3.2.1. However, the interconnection of the components and the progression of time also need to be facilitated. This is the responsibility of the `SimulationKernel` which is explained in Section 3.2.2.

3.2.1 Components

The simulator consists of several components that can be dynamically combined and configured to model a particular experimental setup. This approach enables flexibility both in modeling new setups and in extending the simulator with new components.

Each component implements its own initialization procedure. The order of the initialization steps is enforced, simplifying interdependent initialization involving more than one component. After each time step, all components get notified. Finally, the `finalize` procedure of each component is called at the end of the simulation. A component may choose to implement only a subset of these functions.

3.2.2 Simulation Kernel

The central controller of the simulation is the `SimulationKernel`, which manages the progression of time in the main simulation loop. This loop splits the otherwise continuous molecule movements into steps, creating a discrete-event simulation. Every time step covers a pre-defined duration, usually less than a second.

At the start of the simulation, the kernel initializes the simulation components. This allows enforcing the execution order of the initialization steps. The components

get notified based on events during a time step by the kernel. Finally, the kernel also finishes the simulation by finalizing each component.

In addition to this time progression, the kernel is also responsible for the interconnection of the components. This allows simulation components themselves to be loosely coupled and interchangeable.

3.3 Molecule

The centerpiece of molecular communication is the molecule. It acts as the information carrier in the system by moving from one place to another.

For our purposes, a molecule is simply the internal simulation representation of the information carriers in the real world. These are not confined to individual chemical molecules, but could be made up of liquid droplets, bacteria, nano-robots, or bigger engineered particles. In the Erlangen testbed, the SPIONs correspond to the molecules of this simulation.

Simulating every single information carrier with a molecule is usually not feasible, since there is a very large number of them. The basic assumption made is that information carriers that are close to each other will behave in the same way. This allows them to be represented by a single molecule instead. If a certain, moderately large number of molecules is simulated, the simulation is therefore expected to give a result that is close to a simulation which considers the behavior of all individual information carriers. This enables simulation with only a tiny fraction of the computational resources necessary for the entire, fine-grained simulation on a particle level. This has to be accounted for, for instance when the sensor response is calculated. Additionally, the number of molecules needs to be large enough so that the randomization in their initial positions creates an approximately uniform coverage.

The `Molecule` class in the simulator implements this. It primarily consists of the molecule position, a 3-dimensional vector. This position is in world coordinates, which is transformed to and from local coordinates where necessary. In addition, every `Molecule` is given a unique identifier (ID). This enables tracking of a single `Molecule` throughout the simulation. The `Molecule` can readily be extended with other characteristics, for example to model different information carrier types. Only a single type of SPIONs is used in the Erlangen testbed, so this was not implemented. Other information used by the various simulation components is saved as well, but will covered in the Section of the respective component.

3.4 Channel

The channel essentially represents those simulation components tasked with predicting the behavior of molecules. The `MovementPredictor` (see Section 3.4.3) integrates the molecule movement based on the flow. It retrieves the flow from the `SceneManager` (3.4.7), which forwards the request to the `Object` (3.4.9) that the molecule is in. The object itself contains a `VectorFieldManager` (3.4.5), which is responsible for the interpolation of the flow. It uses a k-d tree to store the actual vector field from which it retrieves the closest cell centers.

3.4.1 Scene System

One limitation of the vector field simulation as presented before is the mesh size. This entire mesh has to be kept in memory during the setup of the CFD simulation and a k-d tree containing the cell centers is stored in memory. With meshes routinely consisting of several million cells, this can become a problem. Additionally, the simulation duration of a CFD simulation tends to scale linearly with the mesh size as well.

I propose to use a "scene" approach to address these issues. This idea is taken from computer graphics, where a complex scenery is broken down into individual objects. Such an object, e.g. a tree, can then be reused several times and exists independently from the other objects or the scene itself. In a molecular communication system, such independent objects can be identified, too. Figure 3.2 illustrates this scene approach. The complex scene has been broken down into two straight pipe objects, two injection objects, and an S-bend. The CFD simulation is run for each object

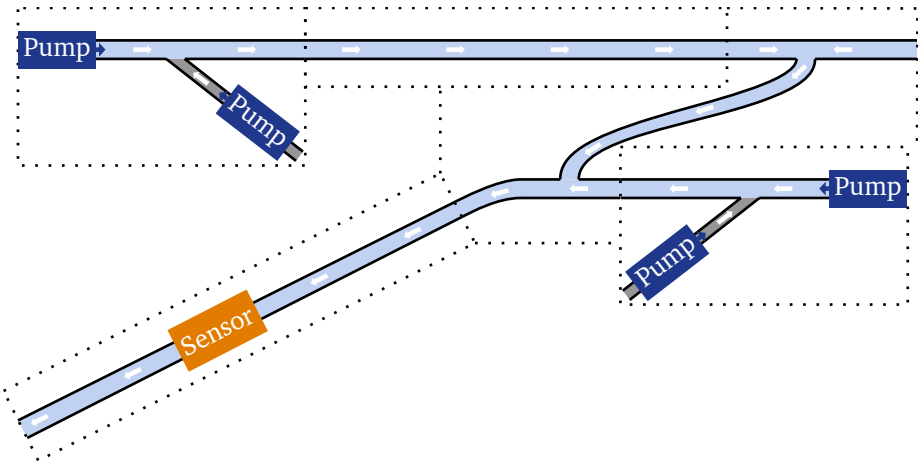


Figure 3.2 – Schematic of a scene with four pumps, a sensor, and five different objects (dashed). Originally created by Lukas Stratmann.

separately, solving the memory issue outlined above. Here, the injection objects and straight pipe segments can be reused. Due to the reuse of objects, the total CFD simulation time is reduced as well.

Unfortunately, the objects in the scene system are not as decoupled from each other as the trees in the graphics example. When a pump turns on or off, the changes in volumetric flow rate need to be propagated downstream. This is why our simulator allows the definition of interconnected inlets and outlets of objects. This approach is illustrated in Figure 3.3 As soon as the flow rate of an outlet changes, the `SceneManager` makes sure to propagate this change to all attached inlets. The object associated with the inlet is notified, calculates its changed outlet flow rates based on the modeled geometry and notifies the `SceneManager`. This recursively updates the flow rates in the entire system. How the inlet flow rates influence the outlet flow rates is implemented by each object. For the Y-piece shown at the left of the Figure, the calculation is a simple addition of inlet flows. The tube simply propagates the inlet flow rate as the outlet flow rate. A splitter like the one shown at the right requires more complex calculations to determine the flow rates.

One additional advantage of the scene system is explained with this changing flow rate. This can be explained using the flow rates of the example in Figure 3.2. With four pumps being either switched on or off, there is a total of 2^4 possible system states. This means that in a single CFD simulation, the entire setup has to be run 16 times to account for each state. This obviously scales very poorly for larger systems. In the scene system, however, several states result in the same flow rate at an object. For example, both injectors do not influence each other. That means that the total number of injector CFD simulations is reduced to 2^2. Also, several injector combinations result in the same flow rates in the S-bend and the sensor tube.

3.4.2 Molecule Manager

The `MoleculeManager` is a central component which encapsulates the state of the simulation. It contains a list of all molecules present in the simulation. Whenever a molecule is added, deleted, or changes one of its attributes, this change is propagated to the manager, which updates its local store. Other components retrieve this list of

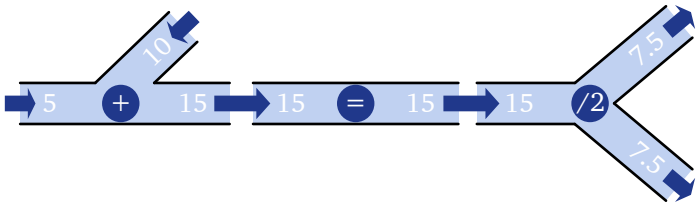


Figure 3.3 – Example flow calculation for three interconnected objects. Volumetric flow rate is given in mL/min

molecules for manipulation, usually by iterating over it and performing operations on each molecule independently. For performance reasons, the changes are usually transferred back to the `MoleculeManager` in the form of an updated list, instead of individual function calls.

3.4.3 Movement Predictor

The particle movement is assumed to match the movement of the fluid at the particle's position. This requires that the simulated flow has a low Stokes number. It would certainly be possible to consider the momentum of the particle, approximate the inertia and simulate other Stokes numbers. For the Erlangen testbed, the nanoparticles are considered sufficiently small and with low inertia. This is why the current implementation of the `MovementPredictor` simply retrieves the flow and uses it to predict the next position.

As pointed out in Section 2.4, there are several approaches of how to determine the new position. Implementing Euler's method is straightforward. The list of molecules is retrieved from the `MoleculeManager` and iterated over. Then, the movement of every molecule is independently predicted. First, the flow at the current position of each molecule is retrieved and saved in $k1$. Then, this flow is multiplied with the time step duration to determine the movement during the time step. This movement is added on top of the current position and returned.

For Runge-Kutta 4, the calculation does not stop there. The flow is retrieved at the centerpoint between the current position and $k1$, multiplied with the time step duration, and saved in $k2$. This process is repeated for $k3$ and $k4$ according to the Equations given in Section 2.4. Finally, $k1$ to $k4$ are weighted, summed up and added to the current position. This is then returned as the new position.

Parallelizing this prediction is easily possible, provided that the flow lookups can be made in parallel as well. However, Python's multithreading support was found to be unsuitable for this. One alternative would be to use multiple processes, which was also tested. A naive implementation with the multiprocessing `map` function provided by Python needed to transfer the vector field across processor boundaries. Therefore, the effective performance of the simulator degraded instead. For these reasons, the simulator only uses a single processor and thread.

3.4.4 Vector Field Parser

For parsing the vector field, the Python library "ofpp"[4] is used. It supports reading OpenFOAM mesh and output files, both in binary and in text output formats. However, some minor errors surfaced when the library was integrated into the simulator. These

[4] `https://github.com/dayigu/ofpp`

errors were addressed and a fixed version was published to the Python Package Index under the new name "OpenFOAMparser"[5]. The parsed data is then transformed into the vector field representation used in the simulator. This data structure consists of two lists of 3-Dimensional vectors. The first list stores the positions of the cell centers of the mesh. The second list stores the associated flow of the cells.

3.4.5 Vector Field Manager

The `VectorFieldManager` abstracts away the access to the vector field returned by the `VectorFieldParser`. As outlined in Section 2.5, some kind of interpolation needs to be applied when determining the flow at the position of a molecule. This lookup needs to be run for every molecule in every time step, so it is paramount that it is optimized. The `VectorFieldManager` implements this interpolation, returning the interpolated flow to the `Object`.

It uses the k-d tree presented in Section 2.3.3 to store the cell centers internally, instead of the list structure used by the `VectorFieldParser`. This tree structure allows the efficient retrieval of the n closest cell centers to any point. The interpolation algorithms I implemented make use of this data structure to improve the simulation speed. These algorithms, nearest neighbor interpolation, Shepard's interpolation, and a modified version of it, are covered in the following sections.

3.4.5.1 Nearest Neighbor Interpolation

When applying the current cell lookup method for nearest neighbor interpolation, it is necessary to determine the exact cell a molecule is in. It would be necessary to determine the cell faces of surrounding cells, their positions, and their orientation relative to the sample point for this. Such an implementation is expected to consume a lot of computational resources just for this lookup. An approximation with higher efficiency is used instead. The single closest cell center, which is not necessarily the center of the cell the sample point is in, is determined. For a regular grid, this approximation method is equivalent to the actual cell center calculation, while still giving appropriate results for an irregular one. It is also fast, since the nearest cell center can be determined efficiently with an optimized spatial data structure like a k-d tree. Using a spatial data structure instead of a simple list of points was therefore one of the earliest optimizations and improved the run time of the simulations by several orders of magnitude.

[5] `https://pypi.org/project/openfoamparser/`

3.4.5.2 Shepard's Interpolation

Using more advanced interpolation algorithms aims to reduce grouping of molecules and creating a smooth channel impulse response. Shepard's interpolation is described to be very smooth and easy to implement. It calculates the distance of each known data point to the sampling point and weights the function value at the data point with the inverse of this distance. The definition of Shepard's interpolation is given in Section 2.5. The simulator implements this by iterating over all cell centers, calculating the distance to the sampling point, and using this for the weights. The exponent μ is configurable.

3.4.5.3 Modified Shepard's Interpolation

The original Shepard's interpolation is global and needs to consider every single mesh cell. For the purpose of the simulator, the performance turns out to be unacceptable, since the mesh easily contains millions of cells. Since a k-d tree readily supports efficient queries for the nearest N_w neighbors, I also implemented the local variant by Franke and Nielson [24]. This means using the modified weighting function given in Equation 2.13.

The radius of the sphere R_{w_i} is determined by retrieving the N_w nearest neighbors from the k-d tree and calculating the distance to the one furthest away. Since the weight of positions on the surface of the sphere of influence is zero, no jumps occur and the interpolation is sufficiently smooth. The distance calculations involved can be reused from the k-d tree lookup, improving performance.

The choice for the configurable exponent μ will be discussed in detail in Section 4.2.5.4. A value for N_w has to be determined as well. Considering the eight cell vertices surrounding the query point appears as a reasonable approach for CFD interpolation. As pointed out in Section 2.5, the most distant value is not incorporated into the modified interpolation. I therefore picked $N_w = 9$ to consider all eight nearest neighbors.

One challenge remains: The boundary of the mesh. As soon as a molecule comes close to the boundary or crosses it, the interpolation will fail. This is because suddenly, all of the known flow values are on one side of it, instead of all around in different directions. Under those conditions, Shepard's methods extrapolate instead of interpolate.

Switching the interpolation method at the wall is my proposed solution specialized for fluid simulations. The interpolation algorithm returns no flow at the wall because of the no-slip boundary condition there. For the boundary cell only, the flow is then extrapolated linearly, approximating the boundary layer. This is done by determining the distance between the sample point and the wall as well as the distance between the cell center and the wall. At the wall, the flow is set equal to

none. At the cell center, the flow is equal to the flow of the cell. At the face opposite of the wall, the flow is double the flow of the cell. For cells adjacent to more than one wall, only the closest wall is considered.

3.4.6 Mesh Manager

Since parsing a vector field and setting up the required data structures is computationally expensive, the `MeshManager` was implemented to use caching wherever possible. Objects can pass an OpenFOAM path to be parsed, which the `MeshManager` instructs the `VectorFieldParser` to turn into an in-memory representation of the cell center positions and their associated flow. A version of this in-memory representation is kept by the `MeshManager`. If the same vector field is used afterwards, only the in-memory version is returned and the parsing is not re-executed. An additional caching step speeds up the initialization of subsequent simulation runs. After parsing, the in-memory representation is serialized with the Python tool "pickle" and stored on disk. Before running the `VectorFieldParser`, the cache folder is checked for existing "pickled" versions of the data. Again, if a cached file is found, the file is deserialized instead and no parsing takes place. Since the CFD simulations only change rarely, this caching can be very useful. When a CFD simulation change occurs, the cache folder needs to be cleared.

3.4.7 Scene Manager

The scene system presented in Section 3.4.1 is implemented with the use of the `SceneManager`.

When implementing the scene system, a decision has to be made between optimizing for memory footprint or simulation run-time. Since the same object can exist several times in the scene at different places, the handling of coordinate transformations for these positions has to be decided. To reduce the memory footprint of the simulator, saving the mesh only once would be a possible solution. This would require transforming from world into object coordinates for every single mesh access, increasing the simulation run-time. Another option would be to load several copies of the mesh into memory and applying the different transformations to the respective meshes only once at the beginning of the simulation. The choice currently implemented in the Pogona simulator is coordinate transformation for each access. This decision was motivated by the scarcity of simulation machines with large memory, while running simulations for a slightly longer time was acceptable.

3.4.8 Teleporting Sensor

The scene system requires the coupling of several objects to create a single interconnected tube system. This coupling is realized with the `SensorTeleporting` component. Where an outlet of one object is connected to an inlet of another object, the teleporter is placed automatically. For this purpose, each object needs to specify its outlet areas. The teleporter geometry then encompasses this outlet area.

The operation of the teleporter itself is simple. Whenever a molecule enters the outlet area of the source object, its object ID is replaced with the ID of the target object of the interconnection. Subsequent requests for the flow of the molecule's position are then routed to the target object instead of the source object.

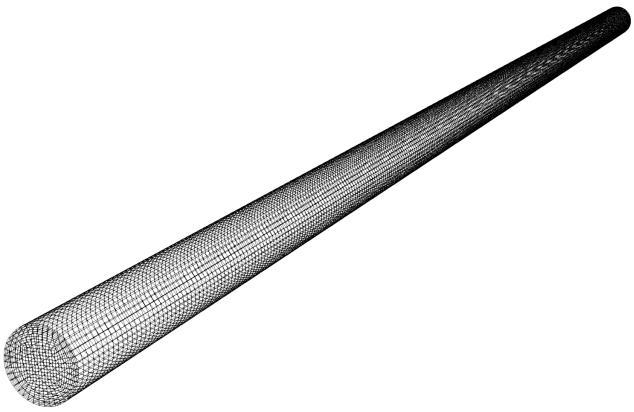
The teleporter is implemented as a Sensor for performance reasons. Since it only needs to work on molecules in the outlet area, it uses the sensor subscription architecture that will be presented in Section 3.6.1 to reduce the simulator run-time.

It has to be ensured that molecules will actually hit the sensor area of the teleporter. Otherwise, they continue in the geometry of the target object, but use the closest flow from the outlet of the source object. The resulting molecule movement would therefore be incorrect. This is why the outlet area needs sufficient depth such that fast moving molecules do not accidentally skip over it. Checking for this is advised when using large time steps.

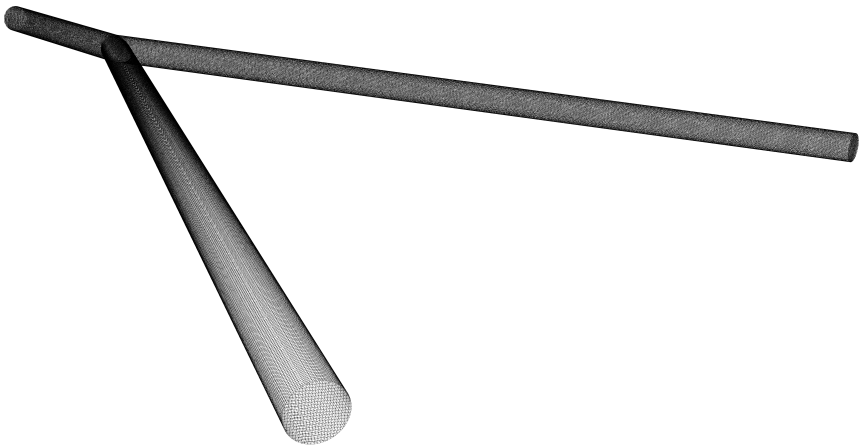
3.4.9 Objects

The scene system presented in Section 3.4.1 supports different `Object` components. These correspond to nearly independent sections of the tube system. For the investigations in this thesis, only two different objects were necessary. These are handled by the `SceneManager`. The first is a simple cylindrical tube, while the second is a Y-piece with tube sections attached to it.

The cylindrical tube is shown in Figure 3.4a. It has a configurable length, outlet zone, and inlet "dead zone". The inlet zone is necessary because the flow profile only develops the proper shape some distance downstream of the inlet. The tube position is translated upstream such that molecules entering will only be positioned at the sections where the flow has properly developed. The outlet zone is the section of the tube where entering molecules will be teleported away to the attached inlet of the next object. The user-configured tube length is achieved by picking the shortest tube mesh that is longer than the sum of the three user-supplied zones. The outlet teleporter is then placed in such a way that the distance from the inlet to the teleporter matches the user-configured length. Thereby, the tube appears shorter to the molecules than the mesh itself would suggest, and the remaining last section of the tube mesh is not actually used.



(a) Tube



(b) Y-piece

Figure 3.4 – Geometries of the simulator objects. Wireframes of the CFD mesh are used for visualization.

The second object, a Y-piece intersection of three tubes, is shown in Figure 3.4b. It does not possess the scaling capabilities of the tube. However, variable-length tubes can be attached to it in the scene if longer tube sections are required. It has an injection inlet, which is the longer tube in the foreground of the Figure. The inlet for the background flow is shown at the top left. Because no molecules enter through here in our scenarios, it has been kept as short as possible. The second long tube at the top right is the outlet of the combined flow. It has a configurable length, again achieved by a variable position of the outlet teleporter. These tubes are positioned such that the injection inlet and the outlet intersect at an angle of 30°, while the other two angles are 165°. This mirrors the Y-piece used in the Erlangen testbed [10].

3.5 Transmitter

The transmission of information is implemented by the spawning of molecules. The basic on-off keying (OOK) modulation works by splitting time into discrete steps. In each step, either information carriers are injected, or not. This spawning of molecules is implemented by the `Injector` covered in Section 3.5.1. The modulation itself is implemented in the `InjectorController` covered in Section 3.5.2, which regulates the `Injector`.

3.5.1 Injector

The creation of molecules is done by an `Injector`. It spawns new Molecule objects and initializes their properties. For example, the initial position of a molecule needs to be set, as well as the object in which it resides. The positions are sampled independently at random from a uniform distribution inside the injector geometry. Possible geometries are cubes, cylinders, spheres, or a single point source. These existing geometries are easily extended to cover more complicated shapes. The object ID a molecule initially receives is determined by the single object each injector is attached to.

An injector may be turned on or turned off. When turned on, it spawns a configurable number of molecules in each time step.

An injector also has an alternative mode of operation, which is called “burst”. There, when the injector is turned on, all molecules are created in a single time step and not continuously. When the injector is turned off again, all molecules that remain in the injection area are deleted. The reasoning for this particular mode of operation can be found in Section 4.1.2.

3.5.2 Injector Controller

The spawning of the `Injector` is toggled by an `InjectorController`, which implements the modulation scheme used by the transmitter. For our investigations, a simple repetitive on-off pattern with constant pauses and injection durations is used. When extending the simulator, this component can be coupled with an existing network simulator such as NS-3 or omnet++ to simulate higher network layers. OOK is already implemented in the Pogona simulator and other modulation algorithms can be easily added. Different amplitudes of injections can be implemented by varying the number of molecules created by the `Injector`.

3.6 Receiver

One fundamental issue is the performance of the simulator for a large number of receivers. A naive implementation requires every receiver to evaluate the position of every molecule at every time step to determine the sensor response. In the following, a more reasonable approach for this problem is presented in Section 3.6.1. How the receivers themselves evaluate the positions is covered in Section 3.6.2. It is used by the simple counting sensor presented in Section 3.6.2.1. A more advanced sensor which weights molecules based on empirical measurements is explained in Section 3.6.2.2. The destructing sensor is used after molecules pass through the other sensors and covered in Section 3.6.2.3.

3.6.1 Sensor Manager

The naive implementation needs to notify all sensors for each molecule movement. Each sensor then evaluates the molecule position and subsequently decides which action to take, e.g. to increment a counter. However, most of the time a molecule is not in a sensor, or only in a single one. This is inefficient and increases the run-time of the simulation. Since sensors are assumed to be stationary, a pre-computation can be employed to speed up this process. If one were to split the simulation area into regions, only the sensors that overlap with the region the molecule is in would need to be notified. Several spatial data structures that do this exist, but the simulator already uses the CFD mesh as its underlying spatial structure. Whenever movement is calculated, the closest cell is already determined to retrieve the associated flow. These cell IDs can then be used to save whether or not a sensor might be interested in a particular molecule. Which sensor overlaps which cell needs to be determined only once in a pre-processing step and can be used in each subsequent time step.

The `SensorManager` component implements this approach. However, it remains optional, both on a global and on a per-object level. Disabling it globally might be necessary because of memory constraints, since the subscription array may become large. Disabling it for a single object might be necessary because the mesh of the object is not suitable to be used for this purpose, or simply because it does not possess one.

For each object in the simulation, the component is keeping a list of mesh cells. Each cell can have several sensors attached to it. During simulation initialization, the `SensorManager` will retrieve the mesh of each object and will iterate over all the cells. For each cell, a sensor subscription is added if the cell center is inside of the sensor area. When a molecule gets a new position from the `MovementPredictor`, the closest cell to the new position is determined. This cell is then used to look up the subscribed sensors for this cell in the list.

3.6.2 Sensors

A receiver consists of a sensor that measures the presence of molecules. In the experimental setup of Unterweger et al. [10], this is a MS2G single frequency susceptibility sensor from Bartington. It consists of a coil around a central cavity in which a material probe is to be placed. Instead of a probe, the plastic tube carrying water background flow and SPIONs is placed through it. Since the particles are superparamagnetic, they increase the magnetic susceptibility measured by the coil.

3.6.2.1 Counting Sensor

A simple assumption would be that the sensor sensitivity is constant along the cavity and that the sensor is not measuring anything outside of the cavity. This is modeled in the following way in the `SensorCounting` component. The section of the tube that is inside the sensor cavity is determined. In each time step, the number of molecules inside this section is determined, while ignoring all other molecules. The sensor response is then simply the number of molecules times a sensitivity factor for the given sensor. The sensitivity factor can be determined both experimentally and analytically. For the experimental sensitivity factor, one can simply scale the output CIR to match the experimental data. For the analytical sensitivity factor, it has to be determined how many real SPIONs one simulated molecule represents. When the magnetic susceptibility of one particle is known, the sensor response is simply the representation ratio times the susceptibility of a single particle.

3.6.2.2 Empirical Sensor

However, the sensor used in the Erlangen experiments exhibits non-linear characteristics. This was determined by Harald Unterweger with the following experiment. A plastic container was filled with the SPIONs. It was then placed in the measurement cavity of the susceptibility sensor, which was oriented vertically. After noting the sensor reading, the container was lifted by 1 mm by inserting a plastic washer below it. This was repeated until the container exited the sensor area at the top of the device.

Figure 3.5 shows the susceptibility readings that were obtained in the experiment. When the container filled the entire sensor cavity, we see that the measured magnetic susceptibility reaches its maximum. When the container has exited the sensor cavity, the magnetic susceptibility comes close to zero. In between, we see non-linear behavior. One can deduce that the sensor is not as sensitive to magnetic particles in the outermost regions of the cavity, but much more sensitive to those in the center where the slope is steeper.

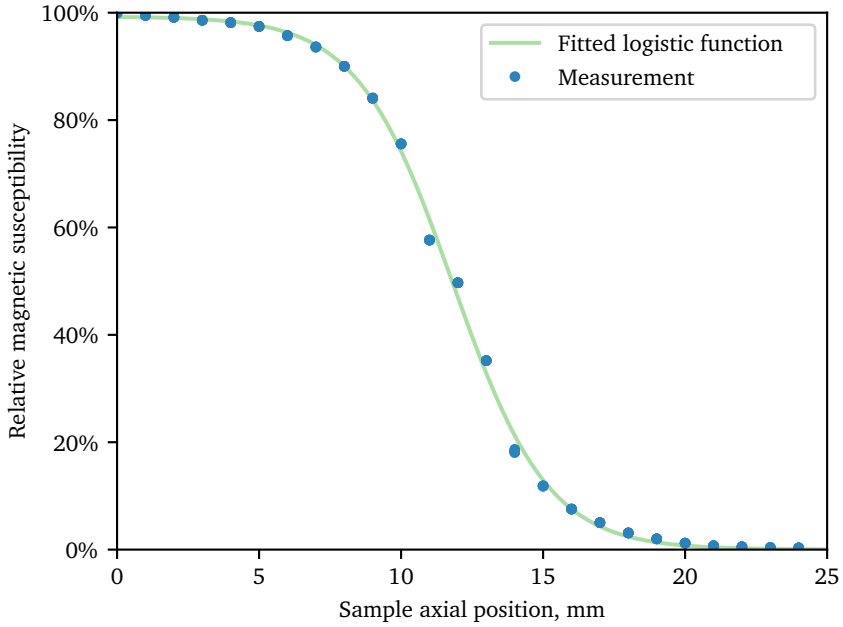


Figure 3.5 – Measured susceptibility of a container filled with SPIONs placed at different offsets in the sensor cavity. Measurement data provided by Harald Unterweger. The logistic function fitted to the measurement is shown.

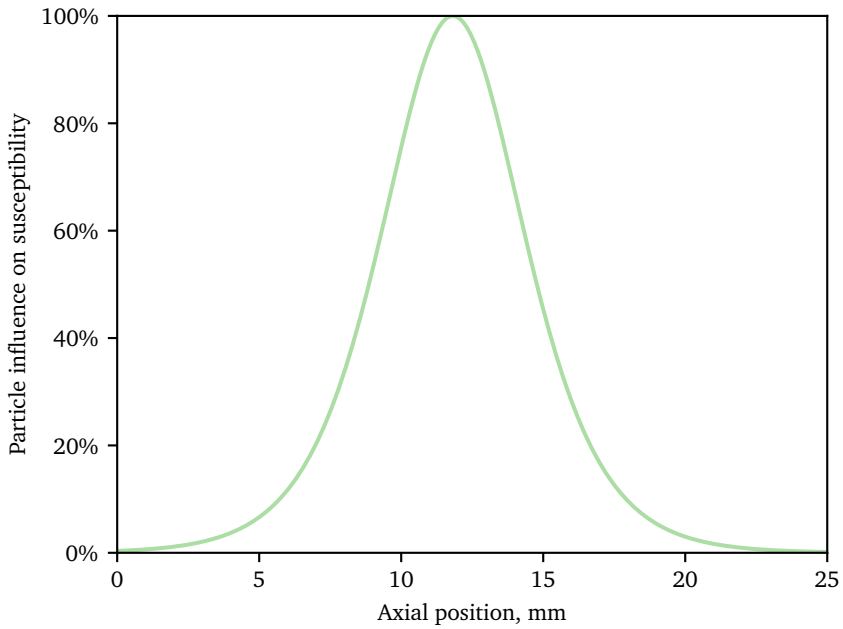


Figure 3.6 – Empirical sensor sensitivity as a function of the nanoparticle offset in the sensor cavity. Obtained by derivation of the fitted logistic function shown in Figure 3.5.

Based on this experiment, one can derive a model that takes this non-linearity into account. First, the measurement errors seen in Figure 3.5 are smoothed out. As Fabian Bronner noted, a logistic function can be fitted nicely to this experiment, returning a smooth and continuous susceptibility prediction. However, the container is inserted step by step, whereas the model needs to predict the sensor response to a single molecule. The container in the measurement therefore acts as an integrator over the real sensitivity function. The underlying sensitivity that produces the observed curve can be determined by derivation of the fitted function. Figure 3.6 shows what the derivative of the logistic function looks like. The function shows a clear central spike where the sensitivity is highest. To both sides, the sensitivity is greatly reduced, with an asymptotic convergence towards zero close to the boundaries of the sensor cavity.

The improved sensor model of the `SensorEmpirical` component thus works in the following way. Again, the molecules outside of the sensor cavity are ignored, since the experiment above confirmed that the sensor sensitivity is very close to zero outside of it. Inside the cavity, the offset of each molecule inside the sensor is calculated. The sensitivity at this offset is used as a factor for the magnetic susceptibility which the single molecule contributes. Again, these sensed magnetic susceptibilities of the molecules are added up to get the total magnetic susceptibility.

As with the linear sensor model, the total amplitude can be scaled based on experiment. However, it is again possible to derive an analytical solution by applying the density of the molecules in the container and its dimensions. Unfortunately, the exact density is not known, so an analytical solution is not easily obtainable.

3.6.2.3 Destructing Sensor

To take care of molecules after sensing, the `SensorDestructing` was created. It simply deletes all molecules that enter its sensor area. This sensor can be placed at the end of pipes where molecules leave the simulated area. For example, the molecules in the Erlangen testbed end up in a waste container after passing through the sensor [10]. Instead of continuing to calculate their movements, which is no longer relevant for the simulation, these molecules can be discarded instead. This boosts the performance of the simulator, since the number of molecules present in the simulation massively influences the simulator run-time. Further details on this can be found in Section 4.3.1.

The decision to implement this component as a sensor was motivated by performance. Since the component is only interested in the molecules in a certain area, e.g. the pipe outlet, it only needs to be informed about molecule movement there. The `SensorManager` already implements the necessary optimizations, so it was used to improve the run-time of the destructor as well.

For the `InjectorController` in burst mode, the `SensorDestructing` is used, too. The `InjectorController` is attached to it and can turn it on and off for these purposes. When the sensor is turned off, it will simply ignore all molecules entering its sensor area.

Chapter 4

Evaluation

The new Pogona simulator needs to be verified and validated. Verification encompasses various checks to ensure the implementation of the underlying theoretic model is correct. Validation determines whether the model actually matches the real world.

Section 4.1 gives the simulator configuration used for these comparisons. Section 4.2 contains a thorough analysis of the correctness of the Pogona simulator. This is followed by an investigation of its efficiency in Section 4.3, since this is one of the primary goals of the simulator. Finally, the simulator output is compared to the Erlangen testbed in Section 4.4.

Parameter	Unit	Value
tube radius	mm	0.75
kinematic viscosity	m^2/s	1e-6
background volumetric flow rate	L/min	5
injection volumetric flow rate	L/min	10
injection volume	µL	17.3
injection molecule count		2000
injection duration	ms	103.8
simulation duration	s	25
time step duration	ms	1
integration method		Runge-Kutta 4
interpolation method		Modified Shepard's, exponent 1
RNG seed		1337

Table 4.1 – Common simulation parameters.

4.1 Reference Cases

In the following, the reference cases used for analysis are covered. This includes the geometry as well as other simulator settings. First, the simulator setup for a simple tube is explained in Section 4.1.1. Then, Section 4.1.2 explains the more complex Y-Piece injection.

The parameters for the reference simulations are listed in Table 4.1. It is explicitly pointed out where a simulation deviates from these.

4.1.1 Simple Tube

The simplest flow-dominated channel for molecular communication is a cylindrical pipe. This has the advantage of having very well understood fluid dynamics and easy to produce experimental setup. The physical channel investigated by Unterweger et al. [10] is a tube, which is why comparison with experimental results is straightforward.

For the CFD simulation, the O-ring mesh shown in Figure 2.2 is used.

The injection in this case is spawning all molecules on a 2-dimensional circular disk at the inlet. This injection happens in burst mode at the very beginning of the simulation. The first time step is the only one where spawning takes place.

The sensor is a simple `SensorCounting` as explained in Section 3.6.2.1. The sensor geometry is a cube which is placed to encompass a section of the tube. Its centerpoint is located 5 cm downstream of the inlet unless stated otherwise. The length of the cylinder is 5 mm, since that is the main sensing area determined by Unterweger et al. [10].

A `SensorDestructing`, as explained in Section 3.6.2.3, is placed immediately behind the counting sensor. It is turned on continuously, deleting molecules as they are about to leave the tube. It is important that this sensor does not overlap with the counting sensor, as the reported count would be too low otherwise. Additionally, this sensor needs to be placed inside the tube, as the optimized sensor subscription feature presented in Section 3.6.1 would malfunction otherwise.

4.1.2 Y-Piece Injection

The Y-piece injection is a more sophisticated scenario that models the complicated interactions between the injection flow and the background flow. In the testbed, the particles are stored in a syringe and injected by a pulsed pump into the main tube. This is modeled by creating the molecules in the inlet of the y-piece object. The injector geometry is thus a cylinder matching the inlet tube. When the injection begins, the injector is turned on in burst mode. Simultaneously, the injection inlet flow for the Y-piece object is set to the injection flow. This means that a CFD simulation

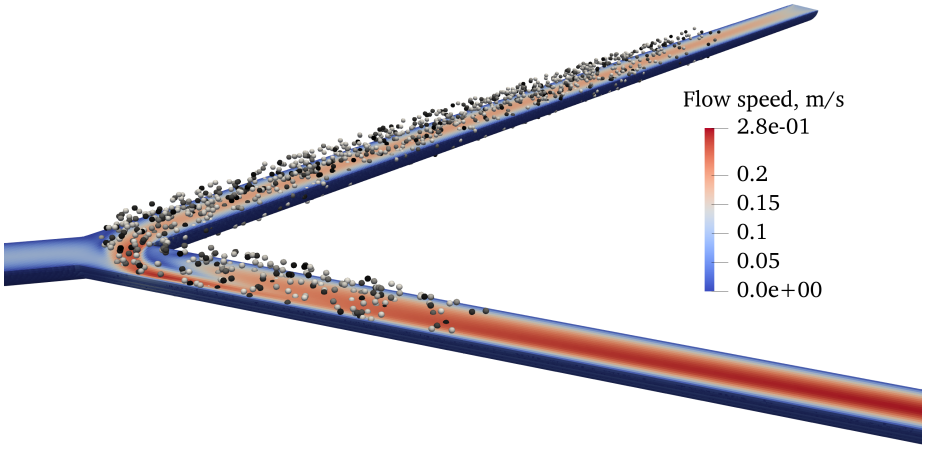


Figure 4.1 – Y-piece during the injection. Molecules are shown as the gray spheres in the top half of the Y-piece. The CFD results are shown in the bottom half, with the flow speed used for coloring.

with both inlets active is used for flow predictions. This carries the molecules in the injection tube around the edge into the outlet tube. After a fixed time has elapsed, the injection finishes. At this time, the inlet flow is set back to none, resulting in another change in the CFD simulation being used. This simulation only has the background flow turned on, and will carry all molecules that made it into the outlet tube further down the tube. At the end of the tube, the sensor is placed. Alternatively, a tube object is placed after the outlet to allow simulation of longer pipes.

One issue with subsequent injections are the molecules that remain in the inlet tube after previous injections. These are only carried into the main pipe when the next injection starts. If they were added to the newly created molecules, they would distort the distribution of molecules in the inlet. Additionally, their movement would have to be calculated even when they remain stationary between the injections. To address this, a destructor is placed with the same geometry as the injector. As soon as the injection finishes, the destructor is turned on. This only deletes molecules that would remain in the inlet otherwise, thus not affecting the CIR.

4.2 Verification

There are different approaches to verify the correctness of a simulator. For a CFD simulation, convergence is the primary concern. This is covered in Section 4.2.1. Since the fluid simulation assumes a laminar flow, this assumption is checked in Section 4.2.2. Whether the CFD simulation properly preserves the configured volumetric flow rates is analyzed in Section 4.2.3. As explained in Section 2.3, the

correct flow profile only develops a certain distance from the inlet of a tube. The unsuitable length of the tube is determined in Section 4.2.4. This section is then configured to be discarded in the Pogona simulator. Because issues were discovered with interpolation, the algorithms are scrutinized in detail in Section 4.2.5. The integration algorithm is verified by comparison with a known-good implementation in Section 4.2.6. A tube object using the analytic solution outlined in Section 2.2 was created as a reference. This eliminates the influence of the CFD simulation and the interpolation algorithm. Section 4.2.7 compares it with the regular CFD-based tube object to quantify the introduced errors.

4.2.1 Convergence

When determining the convergence of CFD simulations, the residuals are investigated first. All simulations used in this thesis were checked for this convergence. The top of Figure 4.2 shows the residuals in one of the tube examples over time. It shows an inverse relationship between the residuals and the simulation duration. As explained in Section 2.3, we see a gradual improvement. However, the solver will stop iterating if the residual error falls below a pre-defined threshold. This can be seen in the bottom of the Figure, where the pressure solver, as well as the flow solvers, reduces the iteration count over time. This eventually leads to an equilibrium, where the both residuals and iterations oscillate. In the current example, this occurs after around 0.6 s. At this point, running the simulation for a longer simulated time will not yield more accurate results.

4.2.2 Turbulence

The CFD model assumes that no turbulence occurs. This is tested by calculating the Reynolds number *Re* for the tube case. We assume a maximum volumetric flow rate of 15 mL/min and a water temperature of 18 °C. We calculate the average flow speed from the volumetric flow rate in Equation 4.1. When inserting this together with the tube parameters into Equation 2.3, we arrive at Equation 4.2.

$$\langle v_z \rangle = \frac{15\,\mathrm{mL/min}}{\pi(0.75\,\mathrm{mm})^2} = \frac{0.25\ \mathrm{cm^3/s}}{1.767 \times 10^{-6}\,\mathrm{m^2}} = 0.1415\,\mathrm{m/s} \tag{4.1}$$

$$\mathrm{Re} = \frac{D\langle v_z \rangle \rho}{\mu} = \frac{0.75\,\mathrm{mm} \times 2 \times 0.1415\,\mathrm{m/s} \times 998.57\,\mathrm{kg/m^3}}{0.001\,051\,8\,\mathrm{N\,s/m^2}} = 201.5 \tag{4.2}$$

The resulting value of 201.5 indicates that this flow does not show any turbulent behavior. Calculating *Re* for the Y-piece is not possible, since no analytical solution

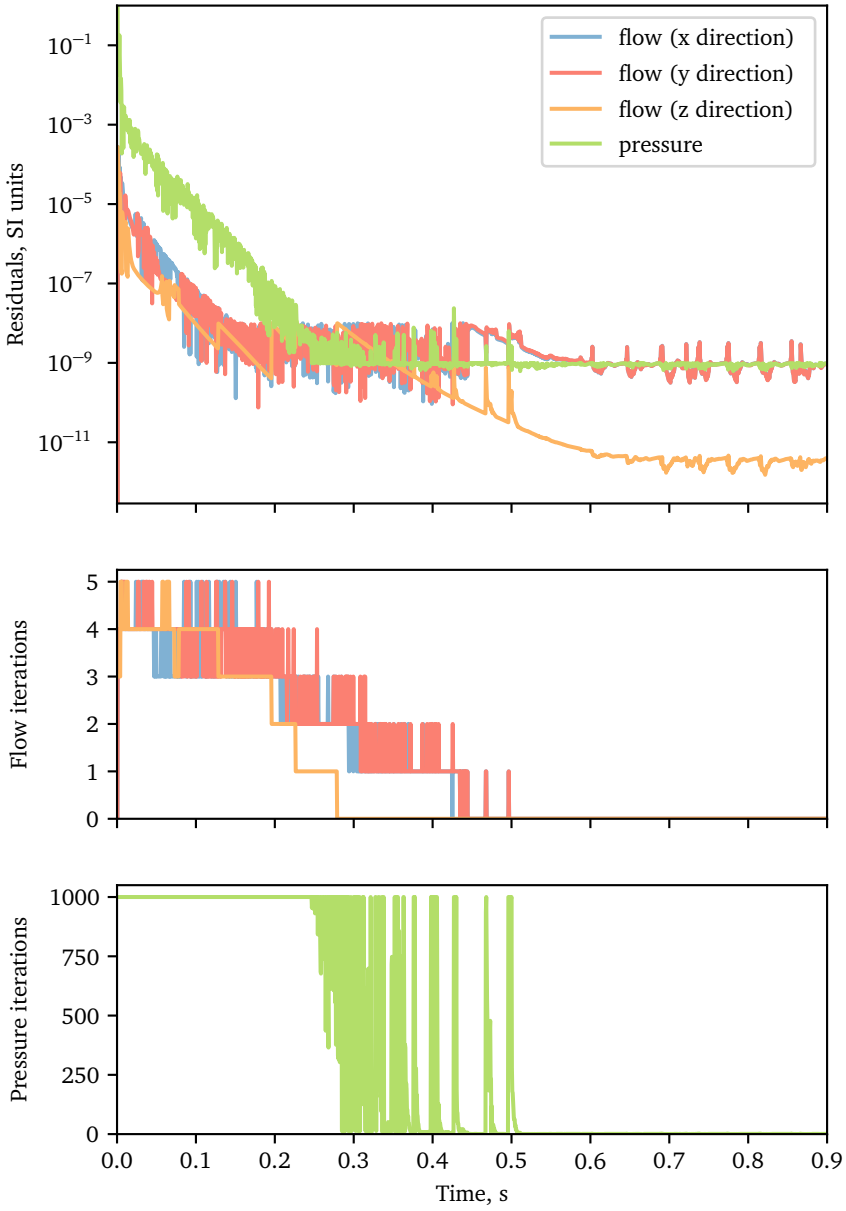


Figure 4.2 – CFD simulation convergence in the 15 cm tube case. Top diagram shows residuals after applying the solver, bottom diagrams show the number of solver iterations necessary.

exists for the flow inside. Because its dimensions are comparable to those of the tube, it is assumed that no turbulence occurs as well.

4.2.3 Flow Conservation

The predicted flow speeds in the tube influence the timing results of the overall CIR simulation.

In order to verify the flow conservation of the calculations, the CFD simulation result is analyzed in paraView. The 15 cm tube was configured with a volumetric flow rate of $2.499 \times 10^{-7}\,\mathrm{m}^3/\mathrm{s}$, which corresponds to 14.994 mL/min. The discrepancy with the intended 15 mL/min was introduced by human rounding errors during simulation setup. A slice of the tube is taken and the flow speed through it integrated, which should return the same value for any position in the pipe. Additionally, this value should match the configured inlet flow rate.

For the snappyHexMesh with a configured flow rate of $2.499 \times 10^{-7}\,\mathrm{m}^3/\mathrm{s}$, the value at both 1 cm and 14 cm distance from the inlet is $2.4989 \times 10^{-7}\,\mathrm{m}^3/\mathrm{s}$. For the regular O-ring mesh, the value at 14 cm is $2.249\,900\,03 \times 10^{-7}\,\mathrm{m}^3/\mathrm{s}$ instead, yielding a smaller deviation from the configured value. Although the mesh structure seems to influence the resulting error, the error magnitude is too small to be noticeable in the context of this thesis. This shows that the basic conservation of flow is being observed by the CFD simulation in the tube case.

For the y-piece, the background inlet is configured with $8.33 \times 10^{-8}\,\mathrm{m}^3/\mathrm{s}$ and the injection inlet is configured with $1.666 \times 10^{-7}\,\mathrm{m}^3/\mathrm{s}$. After both flows join, this should yield a combined volumetric flow rate of $2.499 \times 10^{-7}\,\mathrm{m}^3/\mathrm{s}$. When applying the integration method used for the tube, the flow rate 6 cm downstream from the joining point is $2.498\,998\,899 \times 10^{-7}\,\mathrm{m}^3/\mathrm{s}$. This shows the conservation of flow for the case of confluence.

4.2.4 Flow Profile Development

One issue with CFD simulations of pipes is the inlet area. The common boundary condition at the inlet is a volumetric flow that is constant across the entire area. In this case, the flow speeds at the center will be lower than appropriate, while the flow speeds at the wall will be higher. As the fluid moves further down the pipe, the drag of the wall will slow down the outer layers, while the ones in the center speed up accordingly.

This means that a certain area downstream of the inlet is unusable, since the flow has not fully developed there. Figure 4.3 shows the flow profile at different distances from the inlet in our tube simulation. It is apparent that the flow eventually converges to the theoretically predicted profile at the end of the tube. Determining

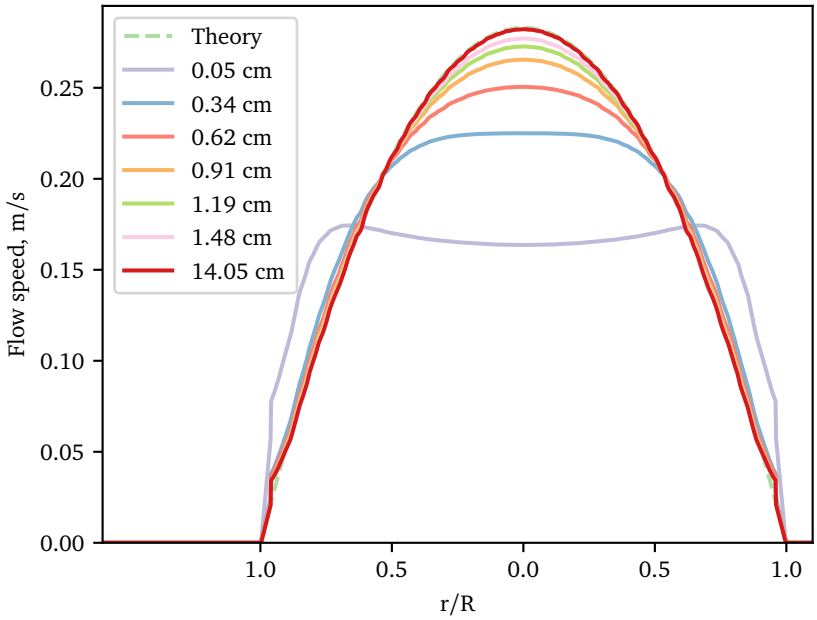


Figure 4.3 – Flow profile at different offsets near the inlet of the tube. Flow profile at the outlet and the theoretic parabola added for reference.

at which point the development is sufficient depends on the accuracy requirements of the simulation.

Table 4.2 shows the flow speed at the center of the pipe. This was determined using the sample utility in paraView. For our purposes, 4 digits of agreement are considered enough. This means that the first 5 cm of the pipe have to be discarded. After that, no significant change to the flow profile is to be expected.

4.2.5 Interpolation

The interpolation algorithms described in Section 3.4.5 turned out to have a large influence on the CIR prediction. To verify it, the first step is to compare the flow profile it produces. Section 4.2.5.1 contains this analysis, concluding that discontinuities

Distance to inlet (cm)	Flow speed (m/s)
1	0.2679
4	0.281752
5	0.28178
10	0.281785
15	0.281785

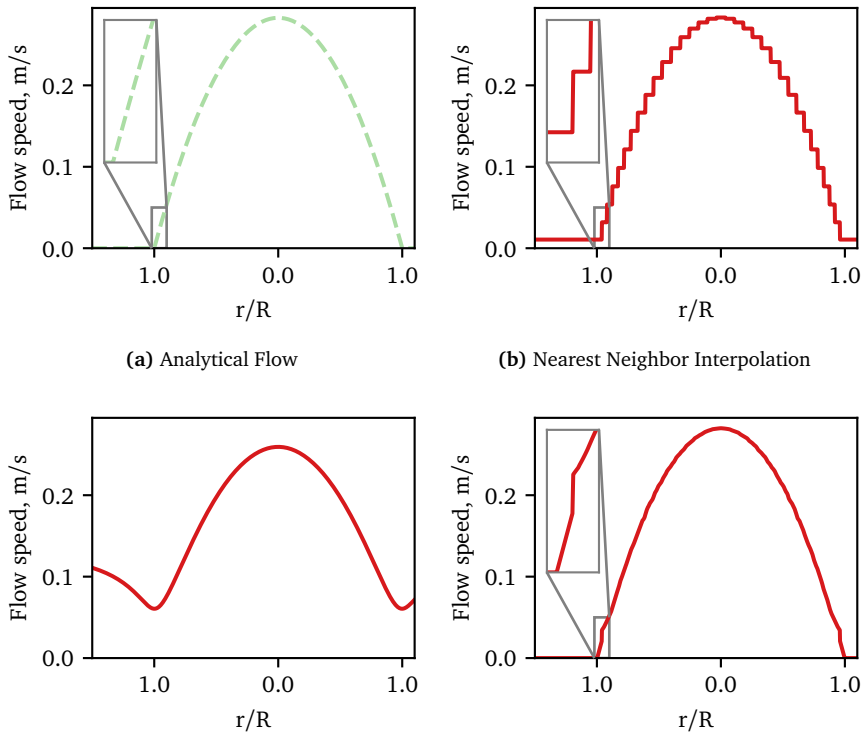
Table 4.2 – CFD Flow speeds at the pipe center.

appear in the flow profile. The influence of these discontinuities for the particle movement is presented in Section 4.2.5.2. The non-uniform particle movement can then be identified in the CIR analyzed in Section 4.2.5.3. To conclude the analysis, the possible IDW exponents are evaluated in Section 4.2.5.4.

4.2.5.1 Flow Profile

Figure 4.4 demonstrates the effect of the interpolation algorithm for an example tube with a flow of 15 mL/min. The flow is sampled in a straight line that cuts through the tube perpendicularly. In particular, the sampling is done near the pipe outlet, 14.05 cm from the inlet. The y-coordinate is fixed to 0 for this purpose, and the x-coordinate is varied in 1500 small steps. The interpolated flow downstream through the tube in z-direction is plotted for each sampling location.

Nearest neighbor interpolation creates a discontinuity at the interface between neighboring cells. This effect is shown as the flow jumps in Figure 4.4b. It becomes apparent that this does not approximate the analytic flow profile well.



(a) Analytical Flow

(b) Nearest Neighbor Interpolation

(c) Shepard's Interpolation

(d) Modified Shepard's Interpolation

Figure 4.4 – Flow profile in a tube, calculated from the CFD simulation using different interpolation algorithms.

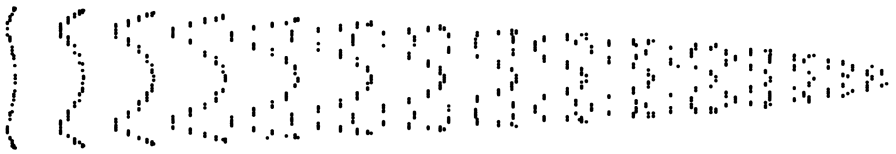
The interpolation result of the original version of Shepard's interpolation is unusable. It uses an exponent of 2, which suffers from the domination of far-away values mentioned in Section 2.5. With a choice of 4, the result is much better, as shown in Figure 4.4c. However, even though the interpolation is very smooth, the differences to the theoretic parabola are simply too big. The extrapolation issues discussed in Section 2.5 are also visible, with the flow speed rising again behind the wall.

Figure 4.4d finally shows the modified version of Shepard's interpolation with an exponent of 1. It can be seen that the interpolant is close to the analytical flow, except for a jump near the wall. At this point, Shepard's interpolation already appears to be suffering from the extrapolation issue. Where the interpolation algorithm finally switches over, a jump appears. The remainder of the interpolant is close to the analytical flow, but there are still inaccuracies left that can be spotted upon close inspection. These will be analyzed in detail in Section 4.2.5.4.

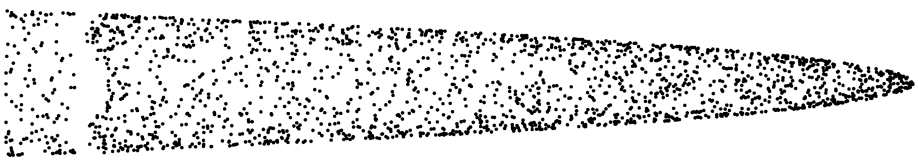
4.2.5.2 Grouping

The effect of the interpolation algorithm choice on the behavior of molecules in the simulator was investigated in the simple tube case. The positions of the molecules are displayed in Figure 4.5.

As expected, molecules move together in distinct groups when nearest neighbor interpolation is used. All molecules within a cell move at the same speed, and thus reach the sensor in groups. This creates large artifacts in the channel impulse response, which now consists of several distinct spikes generated by each group's arrival. Additionally, a lot of groups move at approximately the same speed. This is caused by the O-ring mesh structure shown in Figure 2.2. For the outermost circular



(a) Nearest Neighbor Interpolation



(b) Modified Shepard's Interpolation

Figure 4.5 – Molecule positions after 110 ms in the simple tube case using different interpolation methods.

cell ring, the flow speed of the CFD simulation is virtually identical across all cells due to its symmetry. This is also true for the adjacent cell rings close to the wall. This effect can be observed as the vertical lines to the left of Figure 4.5a. After the four outermost cell rings, the flow speeds of adjacent rings begin to overlap, creating a more uniform distribution of the group speeds.

For the modified Shepard's interpolation, a much more uniform distribution is observed. However, a gap near the inlet remains, where no molecules are present. This can be explained with the change in interpolation algorithm for the outermost cell. The jump in the flow profile shown in Figure 4.4d determines a speed range that none of the molecules reach. Either the speed is below the range because the interpolation algorithm is linear interpolation at the wall, or it is above the range because of the extrapolation issue. This non-existent speed range is then visible as the gap in Figure 4.5b.

4.2.5.3 Channel Impulse Response

The CIR of the nearest neighbor interpolation is not usable at all. The grouping identified in Section 4.2.5.2 is clearly visible. Instead of single molecules entering the sensor area one by one, the CIR displays jumps of groups consisting of up to

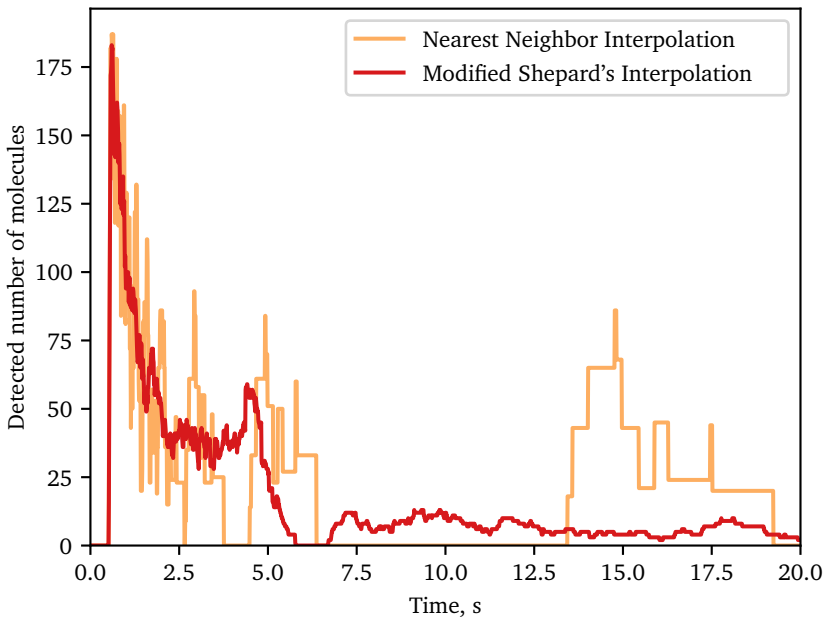


Figure 4.6 – CIR in the simple tube case, using different interpolation algorithms

30 molecules at once. The ring issue is also obvious. After the first 2.5 s, only the molecules from a single ring arrive at the same time, with distinct gaps between the rings. The three outermost rings that are non-overlapping arrive at 3 s, 5 s, and 15 s. They corresponds to the three leftmost vertical lines in the molecule positions in Figure 4.5a. This demonstrates the significant influence of the interpolation algorithm on the CIR.

The molecule-free zone identified in Section 4.2.5.2 has a profound effect on the CIR shown in Figure 4.6. Since the molecules of the It can be seen that when using the modified Shepard's interpolation the susceptibility drops to zero after about 6 s, only to rise again later. In reality, the measured susceptibility should be monotonically decreasing at this time.

4.2.5.4 Exponent of Modified Shepard's Interpolation

As outlined in 2.5, only an exponent of 3 or above is suitable for the original algorithm by Shepard. If the modified local version is used with exponent 4, this results in a less-than-ideal interpolation. The issues can be spotted fairly easily in Figure 4.7, where the interpolated flow is plotted in two dimensions. Although the basic task of creating a mostly smooth interpolation is usually achieved, some discontinuities stand out. For example, at 140.27 mm from the inlet, a band of jumps is visible that

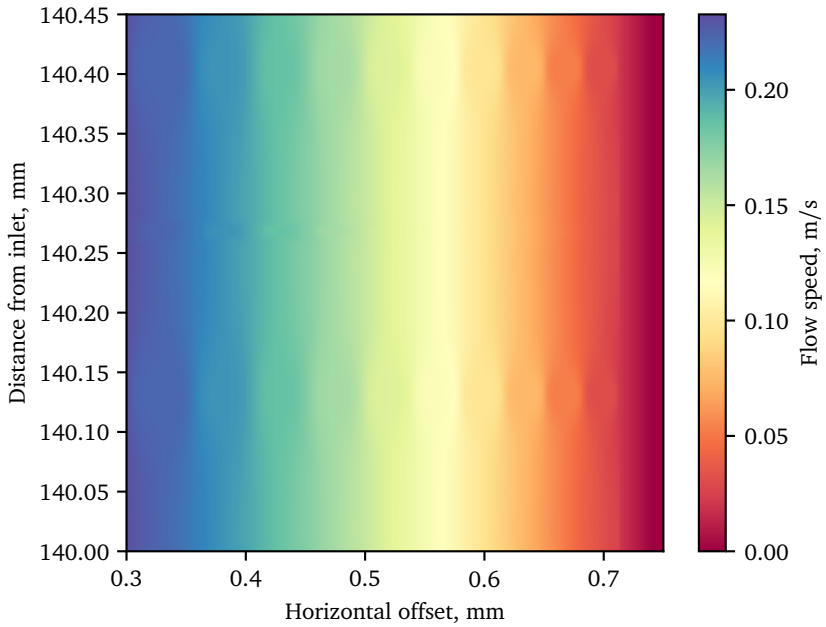


Figure 4.7 – 2D plot of the flow at the wall near the tube outlet, using Modified Shepard interpolation with exponent 4.

goes all across the pipe. This occurs at the border between cells where the closest neighboring cell centers change. Additionally, once the sampling location comes close to the cell centers at 140.13 mm and 140.42 mm, the large exponent causes the center to become dominant over all others. This in turn causes visible artifacts that could be avoided. The last feature to stand out is the perfectly smooth region close to the wall, which is seperated from the remaining area by a discrete jump in the interpolant. This is the effect of the changed interpolation algorithm in the outermost layer of cells. The jump observed in Figure 4.4d thus becomes visible in this plot as well.

Because the issue of far-away values is not applicable in the modified local version, smaller exponents are possible as well. Figure 4.8 shows what influence the exponent has on the resulting CIR. The unexpected peaks change their shape based on the exponent being employed. It is therefore suspected that the peaks are caused by irregularities in the interpolation. For linear weighting with an exponent of one, the peaks are somewhat reduced.

When inspecting the interpolant of linear weighting shown in Figure 4.9, the reason becomes apparent. Whereas an exponent of 4 causes visible banding at the cell border, only small artifacts remain for an exponent of 1. The influence of the cell centers themselves on the surrounding interpolant disappears almost completely.

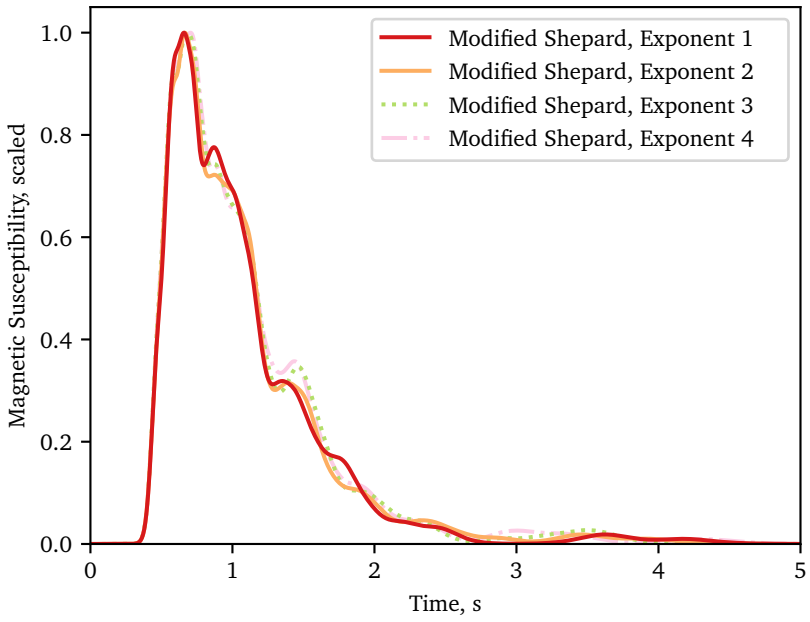


Figure 4.8 – CIR in the Y-piece injection case, using Modified Shepard interpolation with different exponents.

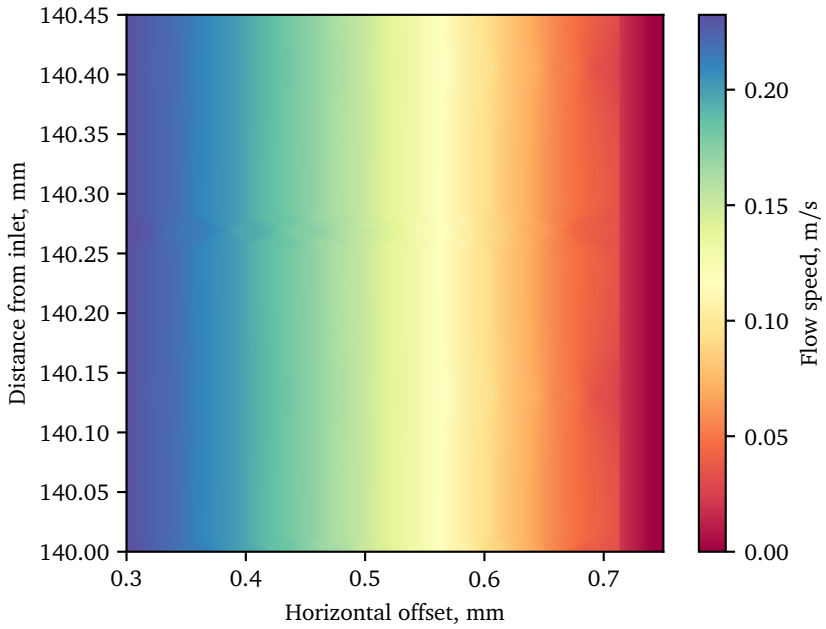


Figure 4.9 – 2D plot of the flow at the wall near the tube outlet, using Modified Shepard interpolation with exponent 1.

Still, horizontal grouping remains in spite of sufficient smoothness. A molecule will travel across thousands of subsequent artifact regions when it moves through the pipe. This means the errors introduced by the artifact will accumulate, which could explain the remaining peaks in the CIR.

4.2.6 Integration

The paraView visualization toolkit[6] provides a variety of data analysis algorithms. It was created by the Sandia National Laboratories, Los Alamos National Laboratories, and Kitware Inc. and is often used for analysis of large data sets, for example at the National Center for Computational Sciences. One of the analysis algorithms is a stream tracer that follows the path of a hypothetical particle through a flow. The flow itself can be imported from OpenFOAM results, through a plugin provided by OpenFOAM. This plugin was used to import the Y-piece CFD results into paraView. Then, a configuration of an injection point source placed in the inlet was created and run in the simulator. The simulator outputs the position of molecules in a format that can be imported in paraView. A screenshot of paraView is shown in Figure 4.10, depicting the molecule positions calculated by the simulator and the ones predicted by the stream tracer. Both the simulator and the stream tracer were configured to

[6] `https://www.paraview.org`

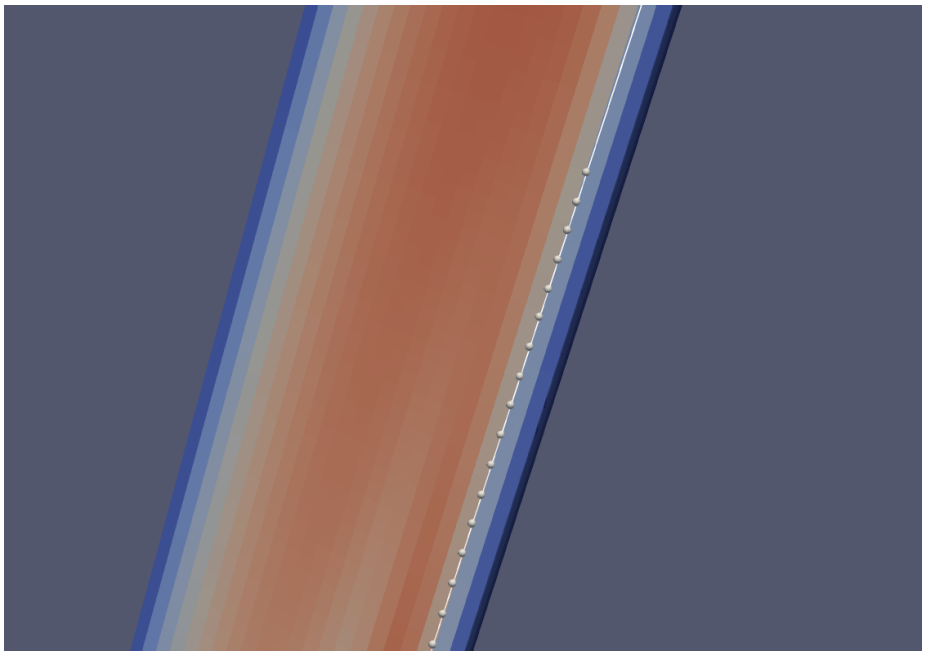


Figure 4.10 – Molecule positions calculated with Runge-Kutta 4 integration. The simulator output is shown as spheres, the paraView stream tracer is shown as the white line.

use the Runge-Kutta 4 integration method presented in Section 2.4. The agreement between the known-good implementation of Runge-Kutta 4 integration in paraView and our simulator confirms the correctness of our Runge-Kutta 4 implementation.

4.2.7 Analytical Tube

The analytical tube is an `Object` implemented as a reference for the regular, CFD-based tube object. Instead of running a CFD simulation, importing it as a vector field and interpolating the flow, it uses a theoretic parabola. The distance to the tube center is determined and input as r into Equation 2.1.

This allows eliminating the influence of the discrete vector field and the interpolation algorithm on the CIR. The resulting CIR for the simple reference case is shown in Figure 4.11. There are three observations to make here. First, the overall shape of the CIR is identical. This means that the CFD simulation, even though discretized and interpolated, results in roughly the same movement prediction as using the analytical formula. Second, there is a spike before about 5 s and drop to zero afterwards that is only observed in the CFD-based simulation. This finally confirms the influence of the interpolation-induced flow profile jump from Figure 4.4d. Third, there is a constant time offset between both curves that is growing. This means that the overall movement speed predicted by the CFD-based tube is slightly lower than

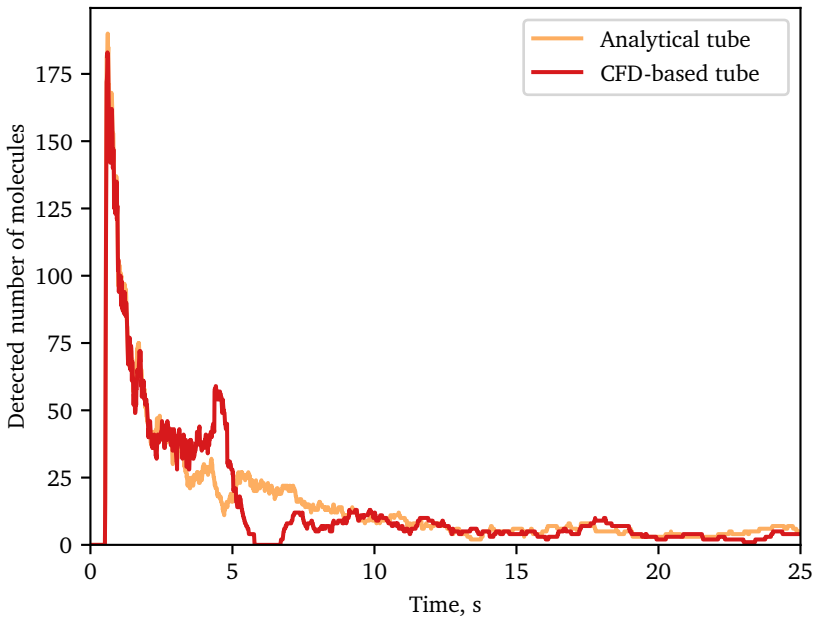


Figure 4.11 – CIR in the simple tube case using either the CFD-based tube object or the analytical tube object.

the analytical one. However, the influence of this discrepancy on the CIR remains small in this case.

4.3 Efficiency

The simulator needs to be efficient in order to be useful. In the following, both the theoretic and the practical computational complexity are analyzed. Section 4.3.1 establishes the theoretic complexity depending on input sizes. The different interpolation algorithms are evaluated empirically in Section 4.3.2.

4.3.1 Theoretic Complexity

The algorithmic complexity of the simulator model presented in Chapter 3 depends on several input sizes, as well as configuration choices.

Most important is the number of molecules m. Since the movement prediction works on each molecule separately, this piece of the simulator exhibits a linear relationship.

The second factor is the time resolution and simulation duration. The simulation kernel goes over each discrete time step in a loop. Therefore, the time complexity is linear in the number of time steps t, which can be calculated with Equation 4.3.

$$t = \frac{\text{simulation duration}}{\text{step duration}} \tag{4.3}$$

Another important input is the CFD vector field, in particularly, the number of cells in the mesh c. The run time of the flow lookup is determined by this input size. In 2.3.3, the result of Friedman, Bentley, and Finkel [19] has been presented regarding the expected run time. Since we are executing a large number of lookups, the average run time of each individual lookup is expected to converge towards this expectation. Additionally, we only need to do a constant number of lookups. We can therefore assume the lookup time scales with $\log(c)$. We arrive at the movement complexity given in Equation 4.4.

$$C_{\text{movement}} = \mathcal{O}\big(tm\log(c)\big) \tag{4.4}$$

The sensor calculations scale in the same way with timesteps and molecules. The naive implementation of notifying all sensors for all molecules in every time step also scales with the number of sensors s. Its complexity is given in Equation 4.5.

$$C_{\text{sensors}} = \mathcal{O}(tms) \tag{4.5}$$

It is apparent that the sensor notification would become the dominating over the flow lookup for a large number of sensors. However, the sensor subscription feature proposed in Section 3.6.1 reduces the computational complexity. For each molecule movement, only the closest cell ID needs to be determined. The complexity for this is equal to Equation 4.4 and can be integrated into the movement prediction, incurring no additional cost. Because the sensor coils cannot physically overlap in the considered testbeds, the assumption that all simulation sensors are non-overlapping appears reasonable. Under this assumption, only one notification at most is necessary for each molecule movement. This notification can be determined in constant time by lookup in the subscription list. Of course, this list now occupies memory of size c. This is acceptable in most cases, since the memory footprint of the movement prediction is proportional to c already. The run time of the sensor calculation is then reduced to Equation 4.6.

$$C_{\text{sensors}} = \mathcal{O}(tm) \tag{4.6}$$

However, the subscription list needs to be initialized as well. This again necessitates checking each mesh cell against each sensor, resulting in the complexity given

in Equation 4.7. This trade-off of increased initialization complexity is preferable in many situations, where scaling with tm is worse than scaling with c.

$$C_{\text{subscription initialization}} = \mathcal{O}(cs) \tag{4.7}$$

Finally, we need to consider the complexity of injecting molecules. The injection itself scales with the total number of molecules. Checking whether an injection is necessary in the current time step scales with both the number of injectors i and the number of time steps. It can thus be described with Equation 4.8.

$$C_{\text{injection}} = \mathcal{O}(m + it) \tag{4.8}$$

The total simulation time complexity of movement, initialization and sensing is given in Equation 4.9. This requires the sensor assumption for Equation 4.6 to hold.

$$C = \mathcal{O}\big(t\big(m\log(c) + i\big) + sc\big) \tag{4.9}$$

4.3.2 Interpolation Benchmarking

To measure the performance of the different interpolation algorithms, a real-world benchmark was executed. The benchmark is similar to the flow sampling from which Figure 4.4 was generated. The execution time of the flow sampling was measured with the Python timeit library. A `VectorFieldManager`, instantiated independently from other simulation components, is used. This eliminates any other possible simulator interactions that might have an influence on the run-time. The underlying tube mesh contained 332750 cells and had a length of 15 cm, which is representative for medium-sized simulation objects. The sampling was done along a y-axis parallel line, starting and ending at the respective tube walls. 1500 samples were taken by accessing the VectorFieldManager directly and saving the returned flow to an array. The timing starts immediately before these samples are taken and ends immediately afterwards. To eliminate noise, the measurements were done on an otherwise idle machine and repeated 10 times. The interpolation algorithm was rotated in a round-robin fashion to ensure that performance degradation over time, e.g. caused by CPU heating, would affect all interpolation algorithms equally.

Interpolation	Mean	Min	Max
Nearest Neighbor	135.5 ms	134.5 ms	137.4 ms
Shepard's	433 098.9 ms	431 127.3 ms	434 675 ms
Modified Shepard's	310 ms	301.8 ms	317.8 ms

Table 4.3 – Execution time of flow sampling at 1500 positions using different interpolation algorithms. 10 repetitions were performed.

The machine used for the benchmark was a dedicated simulation machine. It is equipped with 15.6 GB of RAM and an Intel i7-2600 CPU running at up to 3.40 GHz. The software used is an Ubuntu 18.04.3 operating system with Python version 3.6.8 running in a Python virtual environment. Numpy is used in version 1.16.4, scipy in version 1.3.0.

Table 4.3 shows the measurement results. The minimum and maximum run-time do not deviate significantly from the mean, indicating that the noise in the measurement is low. This was confirmed by checking the 95 % confidence interval.

It can be seen quite clearly that the original Shepard's algorithm is unsuitable due to its excessive run-time of about eight and a half minutes. The k-d tree based local versions perform much better. As expected, the Modified Shepard's algorithm is slower than nearest neighbor, since it needs to look up eight nearest neighbors on top of a single one. However, the resulting run-time is not nine-fold as one might deduce. Due to the workings of the k-d tree lookup, querying for several nearest neighbors only requires a single descent into the region the query point is in. This is why the second lookup is only slightly slower than the first. It also appears that the additional weighting of flow values does not add much run-time either.

4.4 Validation

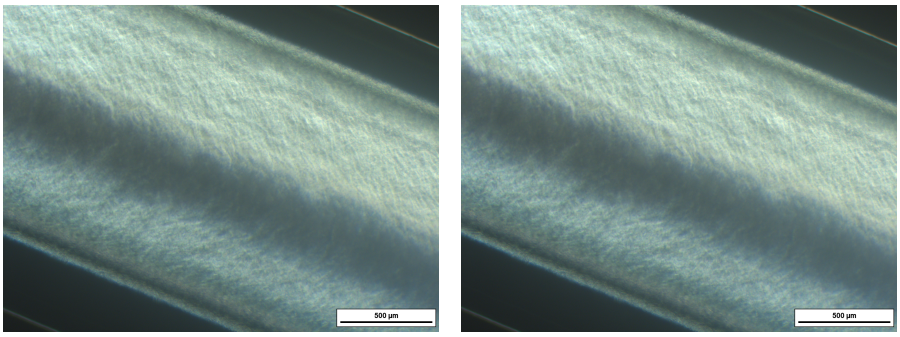
Finally, the model is compared to the real-world testbed in Erlangen. To rule out differences between the simulated and the real diameter of the tubes, microscopy photographs are used in Section 4.4.1. The flow speed of the water inside the tube is determined based on high-speed video recordings in Section 4.4.2. The final two Sections are then concerned with a direct comparison of the CIR as predicted by the Pogona simulator and as measured in the testbed. Section 4.4.3 approaches this from a qualitative angle while Section 4.4.4 applies the metrics from Section 2.6 in a statistical analysis.

4.4.1 Pipe Diameter

The internal pipe diameter has a big influence on the flow speed inside it. This in turn influences the CIR.

To rule out disparities in the CIR, the internal diameter was determined using precision equipment. The vendor gives a diameter of 1.5 mm, which we were able to confirm.

One theory for disparities was the flexibility of the plastic tube used in the Erlangen experiments. Combined with the increased internal pressure caused by the flow, the tube might expand. This was ruled out by Harald Unterweger, who used a microscope to measure the internal diameter at different flow speeds. His



(a) 0 mL/min **(b)** 24.2 mL/min

Figure 4.12 – Microscopy image of tube at different flow speeds. Data provided by Harald Unterweger.

findings are shown in Figure 4.12, where no change in diameter can be detected. Additionally, he used these images to determine an internal diameter of 1.54 mm.

4.4.2 Flow Speeds

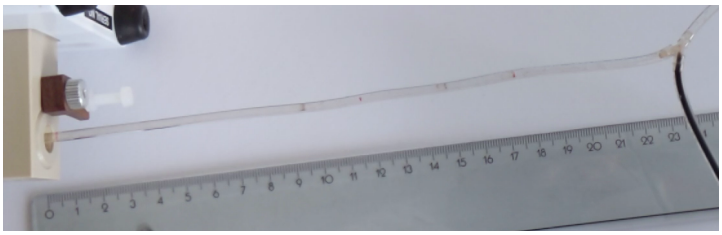
What is known about the flow speed in the experiments is the total volumetric flow rate of each pump, which is configurable. Since the pumps see use in medical applications, these values are assumed to be fairly accurate. By division with the internal area of the tube, the average flow speed can be determined.

To check the results of the flow speeds, a high-speed video of the testbed was created. The testbed was configured with 10 mL/min flow rate for the injection as well as the background. The camera used for this is able to record at 118 frames per second. Selected frames from the recording are shown in Figure 4.13, with a ruler as a scale reference.

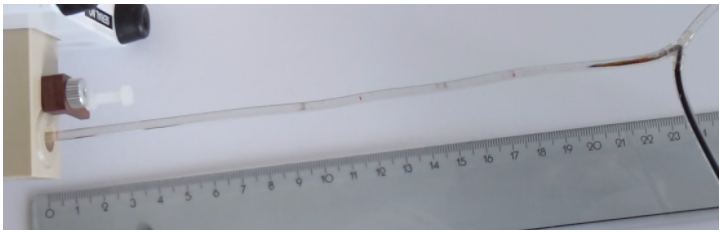
During the injection, which is expected to take about 100 ms, the flow speed is increased, as both the injection flow and the background flow combine. This period corresponds to 11 frames of video, which creates much uncertainty of the precise timing of the injection. The maximum speed of the particle cloud appears to be 0.33 m/s in the first 100 ms. When comparing to the analytic prediction of

Time (ms)	Distance Traveled (mm)
100	33
200	59
300	88
400	106
1000	218

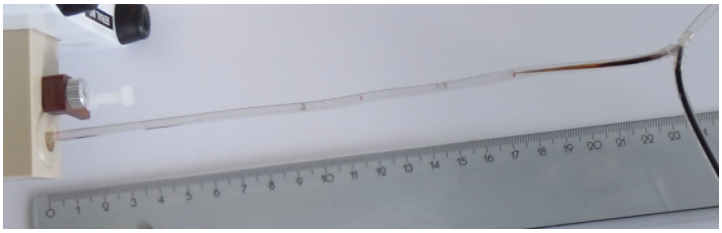
Table 4.4 – Movement of the particle tip in the video shown in Figure 4.13.



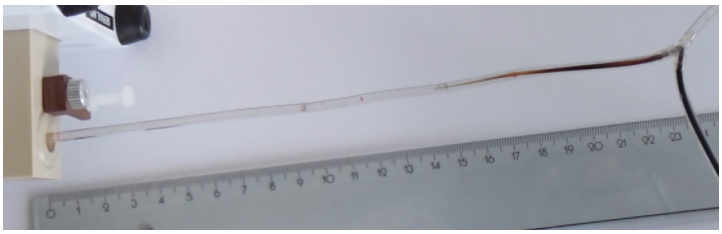
(a) 0 ms



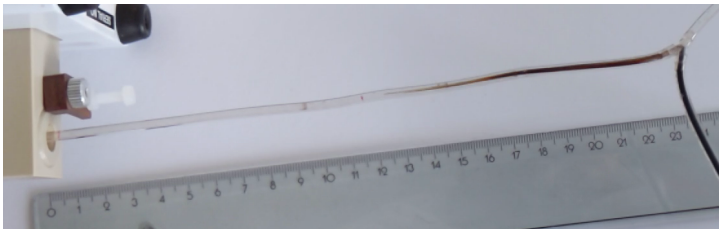
(b) 100 ms



(c) 200 ms



(d) 300 ms



(e) 400 ms

Figure 4.13 – Video frames extracted from a high-speed recording of the Erlangen testbed provided by Harald Unterweger. Times are relative to the beginning of the injection.

a maximum flow speed of 0.3772 m/s for the combined flow of 20 mL/min, this is considerably lower than anticipated.

After the injection, the maximum speed associated with a background flow of 10 mL/min should be 0.189 m/s. However, the particle tip displays a speed of 0.275 m/s between 100 ms and 300 ms. This could be explained by inaccuracies of the injection timing, or the speed-up and slow-down time of the pump.

When considering the traveled distance of the particle tip between 300 ms and 1000 ms, a speed of 0.185 m/s is observed. This speed, unaffected by the injection process, is close to the theoretic expectation.

4.4.3 Qualitative Channel Impulse Response

Figure 4.14 shows a direct comparison between a single simulation injection spike and one from the Erlangen measurements. The measurement has been aligned based on the rising flank and the simulation scaled to match v_{max} of the measurement. The rising flank is considered to be the first data point which rises above 10 % of the peak value.

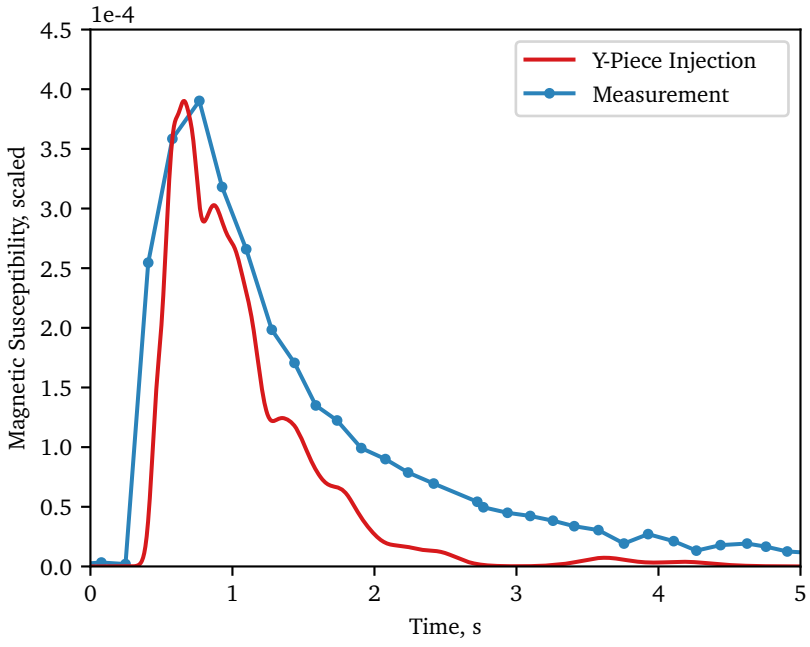


Figure 4.14 – CIR in the Y-Piece injection case compared to the Erlangen measurement. Data provided by Harald Unterweger.

When comparing the measurement and the Y-piece simulation, it becomes apparent that significant differences between them remain, even though the overall shape of the CIR is similar.

The simulation underpredicts the delay spread, especially after the first 1.5 s. After 5 s, no measurable susceptiblity is predicted, but the measurement indicates otherwise. For pulses that are sent shortly after each other, the simulator would therefore predict a lower ISI than observed in the experiment.

Peaks in the simulation appear that the measurement does not show. This stands out at 0.9 s and 1.3 s. Further investigation points to possible inaccuracies in the movement prediction as a root cause, but no definite explanation could be identified.

4.4.4 Channel Impulse Response Metrics

A direct comparison of CIR metrics of measurement and simulation has been created. The measurement consists of 15 consecutive pulse injections at 21 s intervals over a distance of 5 cm. This has been recreated in simulation, using repeated injections in the Y-piece scenario. The pause duration has been chosen to ensure that the pulses do not overlap.

Since the measurement data is unevenly spaced at about 10 Hz, it was first linearly interpolated to the 1 ms sample rate used in the simulation. The peaks were then identified utilizing the scipy python library. Scipy was used to determine the FWHM and delay spread as well. As pointed out in Section 2.6, the delay spread needs to be approximated due to its theoretic infinity. The noise contained in the measurement was analyzed and a maximum noise amplitude of 5 % of V_{max} determined. Consequently, the given delay spread covers the duration of the signal with an amplitude of at least 5 % of V_{max}. Since the noise also has an adverse effect on the RMS delay spread, only the signal section covered by the delay spread is considered for it. For τ_{max}, the same 5 % of V_{max} cutoff is used to determine the rising flank. To enable a fair comparison, the same procedure is applied to the

Metric	Erlangen Testbed		Pogona Simulator	
	Mean	95 % CI	Mean	95 % CI
FWHM	953 ms	867 ms – 1038 ms	578 ms	535 ms – 621 ms
τ_{max}	434 ms	392 ms – 476 ms	260 ms	248 ms – 273 ms
Delay spread	4484 ms	4108 ms – 4860 ms	1782 ms	1746 ms – 1818 ms
RMS delay spread	969 ms	883 ms – 1054 ms	402 ms	393 ms – 411 ms

Table 4.5 – CIR metrics of the Erlangen testbed and the Pogona simulator. The mean and the 95 % Confidence Interval (CI) are given. Measurement data provided by Harald Unterweger.

simulation, even though it does not suffer from noise. For each metric, the mean as well as the 95 % confidence interval over the 15 samples is calculated.

What can not be compared is the propagation delay and V_{max}. The propagation delay for the measurement is not known, since the time of injection for the measurement has not been recorded. V_{max} is currently not being predicted by the simulator. As noted in Section 3.6.2.1 and 3.6.2.2, the reason for this lies in the unknown relationship between a simulated molecule and its measured sensor response.

The results of the comparison are shown in Table 4.5. It is clear that the agreement between measurement and simulation is low. Especially the average delay spread and RMS delay spread differ by more than factor 2 with a probability of more than 95 %. This obvious difference has already been observed in Section 4.4.3. The differences in FWHM and τ_{max} are less pronounced, but still statistically significant. Their average differs by less than a factor of 2 with a probability of more than 95 %. However, the low time resolution of the measurements also needs to be considered. If the rising flank could be identified more accurately than at the provided 100 ms resolution, the true τ_{max} might turn out to be lower.

Chapter 5

Conclusion

This thesis investigated a simulation approach for molecular communication systems. The movement of particles used for communication is predicted on the basis of fluid flow in tube systems, which is calculated with a CFD simulator. The architecture for a simulator using this approach was presented and used for the development of the new “Pogona” simulator prototype. It implements sender and receiver models, supports configurable scenes and utilizes computationally efficient data structures.

The simulator correctness was successfully checked for simplified scenarios with known results. The output of the fluid simulation was verified as well. Additionally, the integration algorithm was compared to a known-good implementation.

For validation purposes, measurement data for an experimental testbed in Erlangen were obtained. The testbed uses magnetic particles injected into a water tube, which are sensed with a susceptometer at the receiver. This testbed was modeled in the simulator and its output compared to the susceptometer data. Low agreement was achieved between simulator output and empirical results. Although the CIR shape matches qualitatively, the predicted delay spreads differ by more than a factor of two from the measurement.

In the future, the prototype developed as part of this thesis can be further improved. Its performance could be increased through multi-threading support. The current Shepard-based interpolation algorithm for the flow already produces good results with only small artifacts. However, it necessarily fails for locations close to the tube wall, which was only partially addressed. Additionally, the cause for the differences in the real-world comparison need to be identified. One issue is the low time resolution of the measurement, which a different sensor could improve upon. Additional forces acting on the particles, such as gravity or diffusion, were not considered in the simulation and might need to be integrated.

Appendices

Appendix A

Shepard Divergence Proof

For IDW, the validity of the interpolation for big datasets is important. In our case, the dataset size easily reaches millions of data points. The influence of data points that are not in the local neighborhood of the query point should not be too large. For the following proof, an infinite, uniformly distributed dataset is assumed. If the the sheer number of non-local data points compared to the limited local data points is not an issue in this case, then the interpolation works for arbitrarily big datasets.

What can be determined is the ratio α between the unnormalized weights of local points and the unnormalized weights of the remainder of the dataset. The local neighborhood is defined to be contained inside an arbitrary, fixed radius r_0, while the remainder is only considered from a distance of r_0 up to a distance of R. This ratio is given in Equation A.1.

$$\alpha = \frac{\sum_{r_0<|x-x_i|<R} W_{\mu,i}(x)}{\sum_{|x-x_i|<r_0} W_{\mu,i}(x)} \tag{A.1}$$

Equation A.2 follows when the original shepard weights given in Equation 2.12 are inserted.

$$\alpha = \frac{\sum_{r_0<|x-x_i|<R} |x-x_i|^{-\mu}}{\sum_{|x-x_i|<r_0} |x-x_i|^{-\mu}} \tag{A.2}$$

Assuming uniform distribution, the summation of the weights of discrete known values x_i can be approximated by integration over the volume of the encompassed space instead. To determine the volume while considering the weight variation with distance, the surface of a sphere with radius r is integrated. The resulting Equation A.3 assumes a dataset with uniform distribution of density ρ.

$$\alpha \approx \frac{\int_{r_0}^{R} 4\pi r^2 \rho r^{-\mu} dr}{\int_{0}^{r_0} 4\pi r^2 \rho r^{-\mu} dr} \tag{A.3}$$

$$= \frac{\int_{r_0}^{R} r^{2-\mu} dr}{\int_{0}^{r_0} r^{2-\mu} dr} \tag{A.4}$$

$$= \frac{\left[\frac{1}{3-\mu} r^{3-\mu}\right]_{r_0}^{R}}{\left[\frac{1}{3-\mu} r^{3-\mu}\right]_{0}^{r_0}} \tag{A.5}$$

$$= \frac{R^{3-\mu}}{{r_0}^{3-\mu}} - 1 \tag{A.6}$$

$$= \left(\frac{R}{r_0}\right)^{3-\mu} - 1 \tag{A.7}$$

When taking the limit for R approaching infinity, the ratio for an infinitely large data set can be determined. It is clear that because of $R > r_o$, the ratio diverges to infinity for $\mu < 3$. Thus, for these exponents, the rest of a large data set is weighted higher than the local neighborhood.

List of Abbreviations

CFD	computational fluid dynamics
CIR	channel impulse response
DNS	direct numerical simulation
FWHM	full width at half maximum
IDW	inverse distance weighting
ISI	inter-symbol interference
MolCom	molecular communication
OOK	on-off keying
RMS	root mean squared
SPION	superparamagnetic iron oxide nanoparticle

List of Figures

List of Tables

Bibliography

[1] L. Felicetti, M. Femminella, and G. Reali, "Simulation of molecular signaling in blood vessels: Software design and application to atherogenesis," *Elsevier Nano Communication Networks*, vol. 4, no. 3, pp. 98–119, Sep. 2013. DOI: 10.1016/j.nancom.2013.06.002.

[2] N. Garralda, I. Llatser, A. Cabellos-Aparicio, and M. Pierobon, "Simulation-based evaluation of the diffusion-based physical channel in molecular nanonetworks," in *30th IEEE Conference on Computer Communications (INFOCOM 2011), Workshop on Molecular and Nano-scale Communication (MoNaCom 2011)*, Shanghai, China: Institute of Electrical and Electronics Engineers, Apr. 2011, pp. 443–448. DOI: 10.1109/INFCOMW.2011.5928854.

[3] E. Gul, B. Atakan, and O. B. Akan, "NanoNS: A nanoscale network simulator framework for molecular communications," *Elsevier Nano Communication Networks*, vol. 1, no. 2, pp. 138–156, Jun. 2010. DOI: 10.1016/j.nancom.2010.08.003.

[4] Y. Jian, B. Krishnaswamy, C. M. Austin, A. O. Bicen, J. E. Perdomo, S. C. Patel, I. F. Akyildiz, C. R. Forest, and R. Sivakumar, "nanoNS3: Simulating Bacterial Molecular Communication Based Nanonetworks in Network Simulator 3," in *3rd ACM International Conference on Nanoscale Computing and Communication (NANOCOM 2016)*, New York City, NY: Association for Computing Machinery, Sep. 2016, 17:1–17:7. DOI: 10.1145/2967446.2967464.

[5] A. Noel, K. C. Cheung, R. Schober, D. Makrakis, and A. Hafid, "Simulating with AcCoRD: Actor-based Communication via Reaction–Diffusion," *Elsevier Nano Communication Networks*, vol. 11, pp. 44–75, Mar. 2017. DOI: 10.1016/j.nancom.2017.02.002.

[6] G. Wei, P. Bogdan, and R. Marculescu, "Efficient Modeling and Simulation of Bacteria-Based Nanonetworks with BNSim," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 12, pp. 868–878, Dec. 2013. DOI: 10.1109/JSAC.2013.SUP2.12130019.

[7] F. Bronner and F. Dressler, "Towards Mastering Complex Particle Movement and Tracking in Molecular Communication Simulation," in *6th ACM International Conference on Nanoscale Computing and Communication (NANOCOM 2019), Poster Session*, Dublin, Ireland: Association for Computing Machinery, Sep. 2019, 36:1–36:2. DOI: 10.1145/3345312.3345490.

[8] J. S. Curtis and B. van Wachem, "Modeling particle-laden flows: A research outlook," *AIChE Journal*, vol. 50, no. 11, pp. 2638–2645, 2004. DOI: 10.1002/aic.10394.

[9] W. Wicke, T. Schwering, A. Ahmadzadeh, V. Jamali, A. Noel, and R. Schober, "Modeling Duct Flow for Molecular Communication," in *IEEE Global Communications Conference (GLOBECOM 2018)*, Abu Dhabi, United Arab Emirates, Dec. 2018, pp. 206–212. DOI: 10.1109/GLOCOM.2018.8647632.

[10] H. Unterweger, J. Kirchner, W. Wicke, A. Ahmadzadeh, D. Ahmed, V. Jamali, C. Alexiou, G. Fischer, and R. Schober, "Experimental Molecular Communication Testbed Based on Magnetic Nanoparticles in Duct Flow," in *19th IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC 2018)*, Kalamata, Greece, Jun. 2018, pp. 1–5. DOI: 10.1109/SPAWC.2018.8446011.

[11] G. Karsenty and E. N. Olson, "Bone and Muscle Endocrine Functions: Unexpected Paradigms of Inter-organ Communication," *Cell*, vol. 164, no. 6, pp. 1248–1256, Mar. 2016. DOI: 10.1016/j.cell.2016.02.043.

[12] N. Farsad, W. Guo, and A. W. Eckford, "Tabletop molecular communication: Text messages through chemical signals," *PLOS ONE*, vol. 8, no. 12, pp. 1–13, Dec. 2013. DOI: 10.1371/journal.pone.0082935.

[13] V. Jamali, A. Ahmadzadeh, W. Wicke, A. Noel, and R. Schober, "Channel Modeling for Diffusive Molecular Communication—A Tutorial Review," *Proceedings of the IEEE*, vol. 107, no. 7, pp. 1256–1301, Jul. 2019. DOI: 10.1109/jproc.2019.2919455.

[14] "Recommended Practice for Nanoscale and Molecular Communication Framework," IEEE, Std 1906.1-2015, Jan. 2016.

[15] R. B. Bird, W. E. Stewart, and E. N. Lightfoot, Transport Phenomena, 2nd ed. New York City, NY: John Wiley & Sons, 2002.

[16] J. H. Ferziger and M. Perić, Computational methods for fluid dynamics, 3rd ed. Berlin, Heidelberg: Springer, 2002.

[17] N. Agrawal, G. H. Choueiri, and B. Hof, "Transition to turbulence in particle laden flows," *Physical Review Letters*, vol. 122, no. 11, p. 114 502, 2019. DOI: 10.1103/PhysRevLett.122.114502.

[18] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[19] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic time," *ACM Trans. Math. Software*, vol. 3, no. SLAC-PUB-1549-REV. 2, pp. 209–226, 1976.

[20] J. C. Butcher, Numerical Methods for Ordinary Differential Equations. Chichester, United Kingdom: John Wiley & Sons, 2008.

[21] C. Runge, "Über die numerische Auflösung von Differentialgleichungen," *Mathematische Annalen*, vol. 46, no. 2, pp. 167–178, 1895.

[22] J. C. Butcher, "A history of Runge-Kutta methods," *Applied numerical mathematics*, vol. 20, no. 3, pp. 247–260, 1996.

[23] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in *23rd ACM national conference*, Las Vegas, NV: Association for Computing Machinery, Aug. 1968, pp. 517–524.

[24] R. Franke and G. Nielson, "Smooth interpolation of large sets of scattered data," *International journal for numerical methods in engineering*, vol. 15, no. 11, pp. 1691–1704, 1980.

[25] J. Wilhelms, J. Challinger, N. Alper, S. Ramamoorthy, and A. Vaziri, "Direct volume rendering of curvilinear volumes," *ACM SIGGRAPH Computer Graphics*, vol. 24, no. 5, pp. 41–47, Nov. 1990. DOI: 10.1145/99308.99318.

[26] A. Goldsmith, Wireless Communications. Cambridge: Cambridge University Press, 2005.