Numerical investigations on the hydrodynamics of bacteria-laden flow in the rotor-stator spinning disc reactor

Jansen, Rick

Award date:
2022

EINDHOVEN
UNIVERSITY OF
TECHNOLOGY

Numerical investigations on the hydrodynamics of bacteria-laden flow in the rotor-stator spinning disc reactor

Department of Chemical Engineering and Chemistry

Sustainable Process Engineering

Author
R. (Rick) Jansen BSc

Eindhoven, April 20, 2022

”Big whirls have little whirls,
which feed on their velocity.
And little whirls have lesser whirls,
and so on to viscosity.”

Lewis F. Richardson

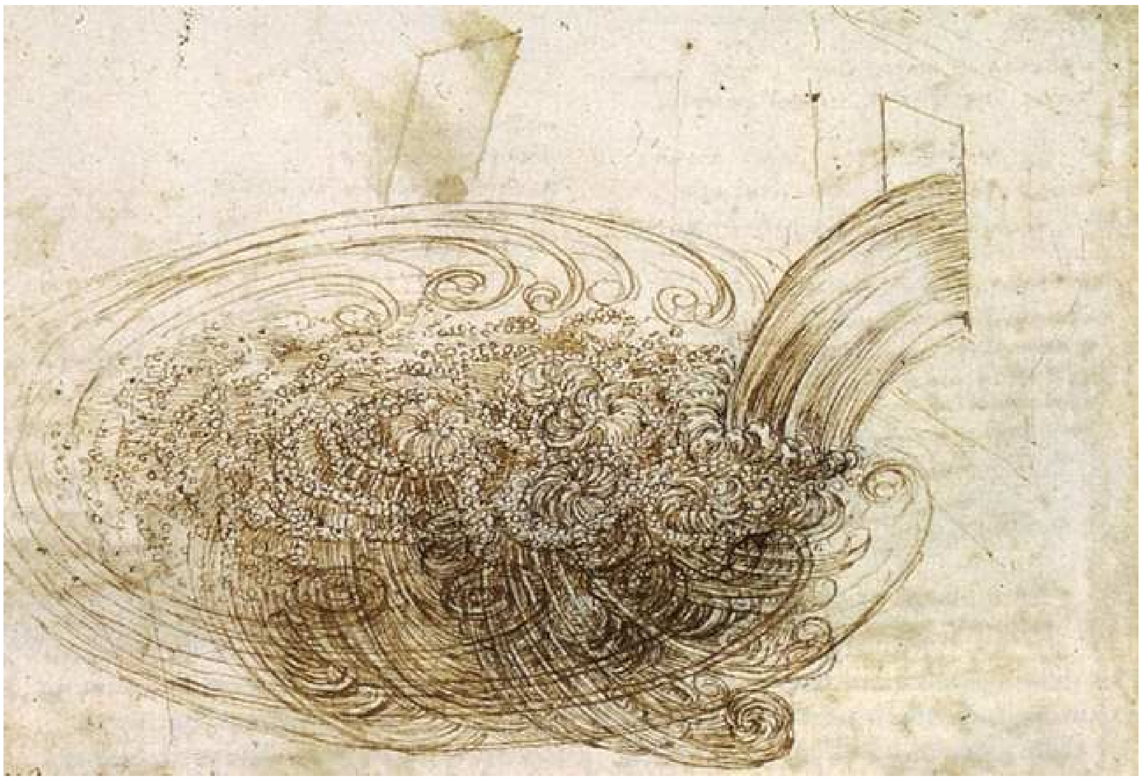


Figure 1: A turbulent cascade as seen by Leonardo da Vinci. Reproduced from [1].

Abstract

Currently, a pathway to transition away from the use of fossil fuels that is widely researched is the conversion of bio-based syngas. Many industrial processes that convert syngas operate on metal-based catalysts under energy-intensive conditions. An alternative method to convert syngas is by means of fermentation by acetogenic bacteria, such as *Clostridium autoethanogenum*. However, the conventional bioreactors are typically mass transfer limited, yielding high reactor volumes. A method to overcome these mass transfer limitations is by utilizing the rotor-stator spinning disc reactor (rs-SDR), which exhibits high mass transfer rates.

In this work, Computational Fluid Dynamics (CFD) simulations are constructed and validated to study the hydrodynamics in the rs-SDR and study the mechanical effects of the hydrodynamics in a turbulent Couette flow on the bacterial cell. For this, Large Eddy Simulations (LES) are employed to capture the turbulent nature of the flow using the wall-adapting local eddy viscosity sub-grid-scale turbulence model. Ultimately, an Eulerian-Lagrangian simulation of bacteria-laden flow in the rs-SDR is constructed using the multiphase paricle-in-cell (MP-PIC) model.

A validation case for a rotor-stator cavity is constructed and CFD simulation results are compared with literature. The simulation results are in good agreement with the literature results for the lower radial positions, but the model deviates more significantly at higher radial positions. The simulation for the rotor-stator cavity is extended to the complete rs-SDR with optional throughflow. Several verification tests are successfully performed on the simulation results. As an experimental validation, the residence time distribution (RTD) is compared with simulated RTDs. For the turbulent regime in the rs-SDR, the correspondence with the experimental RTD is satisfactory.

A turbulent Couette flow is simulated and validated and a bacterial cell is placed in the flow field by means of an Eulerian simulation with the Volume of Fluid method. A tumbling motion is observed. The drag force is found to be the dominant force acting on the bacterial cell, but according to the Tresca and Von Mises failure criteria, plastic deformation in the rs-SDR is not likely.

Finally, the methodology of simulating particle-laden flows in the rs-SDR using an Eulerian-Langrangian simulation is explicated and the particle dynamics are found to be in line with the simulated hydrodynamics.

List of Figures

List of Tables

Listings

Nomenclature

Symbol	Description	Unit
Latin		
$\boldsymbol{a}$	Acceleration	m s^{-2}
c	Proportionality factor in transition radius correlation	-
$\boldsymbol{C}$	Cross stresses	m^2 s^{-2}
C_D	Drag coefficient	-
C_L	Lift coefficient	-
Cq_r	Local throughflow coefficient	-
C_S	Constant in Smagorinsky-Lily turbulence model	-
C_{vm}	Virtual mass coefficient	-
C_W	Constant in WALE turbulence model	-
d	Diameter	m
D_m	Molecular diffusion coefficient	m^2 s^{-1}
D_t	Turbulent diffusion coefficient	m^2 s^{-1}
E	Energy spectrum of turbulence	m^3 s^{-2}
E_{mod}	Residence time distribution curve from the engineering model	s^{-1}
f	Passive scalar	-
F	Force	kg m s^{-2}
g	gravitational acceleration	m s^{-2}
g_{ij}	Deformation tensor	s^{-1}
G	Ratio between axial clearance and disc radius	-
k	Turbulent kinetic energy	m^2 s^{-2}
k_{HB}	Consistency coefficient in Herschel-Bulkley model	m^2 s^{-2}
K	Entrainment coefficient	-
l_{path}	Path length in Beer-Lambert's law	m
L	Domain length	m
$\mathbf{L}$	Leonard stresses	m^2 s^{-2}
L_0	Integral scale	m
$\mathbf{M}$	Torque	Nm
n	Number of CSTRs in engineering model for the rs-SDR	-
N	Number of counts in spectrometry	-
n_{HB}	Exponent in the Herschel-Bulkley model	-
p	Pressure	kg m^{-1} s^{-2}
P_s	Solid pressure	kg m^{-1} s^{-2}
$\mathbf{R}$	Reynolds stresses	m^2 s^{-2}
r^*_{trans}	Normalised transition radius	-
r_d	Disc radius	m
$\mathbf{r}$	Lever arm vector	m
s	Axial clearance between rotor and stator	m
S_{ij}	Symmetric deformation tensor	s^{-1}
S^d_{ij}	Symmetric deformation tensor in WALE turbulence model	s^{-1}
t	Time	s
t_{eddy}	Eddy turn-over time	s
u	Velocity	m s^{-1}
u_τ	Friction velocity	m s^{-1}

y^+	Normalised wall-normal coordinate	-
Greek		
α	Volume fraction	-
α_{cp}	Close packing volume fraction	-
γ	Parameter in Gaussian filtering function	s^{-2}
$\dot{\gamma}$	Shear rate	s^{-1}
δ	Initial separation from condition in Lyapunov exponent	-
δ_{ij}	Kronecker delta	-
δ_{bl}	Boundary layer thickness	m
Δ	Filter cutoff width	m
ε	Energy dissipation rate	m^2 s^{-3}
ε_m	Molar extinction coefficient	m^2 mol^{-1}
ε_s	Small value in numerical Harris-Crighton model	-
η	Kolmogorov microscale	m
θ	Particle distribution function	-
κ	Eddy wavenumber	m^{-1}
λ	Lyapunov exponent	s^{-1}
μ	Dynamic viscosity	kg m^{-1} s^{-1}
ν	Kinematic viscosity	m^2 s^{-1}
ν_t	Turbulent kinematic eddy viscosity	m^2 s^{-1}
ν_{eff}	Effective turbulent kinematic eddy viscosity	m^2 s^{-1}
ν_0	Viscosity at low shear rates	m^2 s^{-1}
ρ	Density	kg m^{-3}
τ	Stress tensor	kg m^{-1} s^{-1}
τ_0	Density normalised yield stress	m^2
τ_p	Particle relaxation time	s
τ_K	Kolmogorov time	s
τ_e	Eddy turnover time	s
τ_m	Mean residence time	s
τ_{PF}	Residence time of plug flow zone	s
ϕ	Sphericity	-
ϕ_v	Volumetric flow rate	m^3 s^{-1}
ω	Turbulence frequency	s^{-1}
ω_{rot}	Rotational speed	s^{-1}
Subscripts		
SGS	Sub-grid-scale	-
Superscripts		
$+$	Normalised	-
Operators		
$\overline{X}$	Ensemble averaged quantity	
X'	Fluctuating part of quantity	
$\tilde{X}$	Spatially filtered quantity	
G	Filtering function	
∇_x	Divergence with respect to position	
∇_v	Divergence with respect to velocity	
Dimensionless numbers		
Co	Courant-Friedrichs-Lewy number	-
Pe	Péclet number	-
Re	Reynolds number	-
Re_h	Reynolds number with respect to axial gap	-
Re_ω	Rotational Reynolds number	-

Re_w	Reynolds number with respect to vorticity	-
Sc_t	Turbulent Schmidt number	-

Contents

Chapter 1
Introduction

With the ever-growing needs of the global population, the urgency of efficient use of energy and resources has become more prominent [2]. With the current absence of abundant carbon-neutral energy resources, an increase in energy consumption is prone to an increase of emission of greenhouse gases. The chemical industry has a major role in the emission of greenhouse gases and energy usage [3], which comes with great responsibility in transitioning towards a more environmentally sustainable industry. One technique to reduce the emission of greenhouse gases is by transitioning from fossil fuels to fuels from lignocellulosic biomass [4]. This entails the production of syngas from gasification of biomass. Thereafter, the syngas can be chemically converted towards valuable fuels or feedstock for the chemical industry.

Industrial processes that currently convert syngas into more valuable products consist of metal-based catalytic processes that operate under energy-intensive conditions [5]. Chemical conversion routes under investigation consist of catalytic ethanol production [6] and hydrogenation to methanol [7], amongst others . Another widely researched syngas conversion route is the hydrogenation to hydrocarbons through the Fischer-Tropsch mechanism [8].

In contrast to the metal-based catalytic processes, microbial pathways of converting syngas into valuable chemicals are currently researched [9]. Microbial syngas fermentation can produce value-added chemicals without the need of catalysts or solvents and sustainably reduce the pressure on biomass and land use [9–11]. Additionally, significantly higher selectivity can be obtained in biochemical conversion pathways [12]. The use of anaerobic syngas fermenting microbes reduces the competition in the food market, which is viewed as a major drawback in first generation biofuels [12]. A microbial syngas fermentation pathway that is promising is the fermentation by *Clostridium autoethanogenum*, which converts CO or CO_2 and H_2 into acetic acid, ethanol and 2,3-butanediol [13].

The standard reactor set-up for microbial syngas fermentation to acetate and ethanol is limited to (fed) batch reactors, continuously stirred tank reactors and bubble columns [14]. However, it is known that the volumetric mass transfer rate from gas to liquid is typically limited in these reactor set-ups. The rotor-stator spinning disc reactor (rs-SDR) is a reactor that can overcome mass transfer limitations by intensified mass transfer due to high shear rates induced by the rotor [15–17], yielding mass transfer coefficient up to 40 times higher than in conventional equipment [15].

Before the fermentation process by *Clostridium autoethanogenum* can be intensified by operation in the rs-SDR, research should be performed on the general hydrodynamics in the rs-SDR. Numerous studies have been performed on the hydrodynamics in the rotor-stator spinning disc reactor [18, 19], yet one inclusive model is still absent. Computational fluid dynamics (CFD) can be used to visualise all important flow and turbulence aspects and contribute to such inclusive model. Additionally, the effect of the shear forces and turbulent fluctuations in the reactor on the bacteria is still topic for research. To study these effects, both reactor scale as bacterial scale CFD models will be constructed and validated.

In Chapter 2, an extensive explanation of theoretical concepts is provided, which ranges from the essence of turbulence, hydrodynamics in the rs-SDR and the application of computational fluid dynamics in turbulent flows to the bacterial syngas fermentation by *Clostridium autoethanogenum* and bacterial cell wall deformation. In Chapter 3, the methodology that was used to construct the computational fluid dynamics models are treated. In Chapter 4, the results of the models are discussed and a conclusion is drawn from these results in Chapter 5. This thesis is concluded with recommendations for further study in Chapter 6.

Chapter 2
Theoretical Background

2.1 Turbulence

Several definitions of turbulence are available and many try to catch the complex behaviour of turbulent flow in a single definition. It turns out that giving a formal definition of turbulence is rather difficult, and it may not be very useful to provide one [20]. A better way to define turbulence is by characterizing the features of flow that exemplify the turbulent behaviour.

At a critical value of the Reynolds number ($Re = uL\nu^{-1}$), which is a measure for the relative importance of inertial forces and the viscous forces, the inertial forces dominate the viscous forces, initiating a radical change towards turbulent flow [21]. Turbulent flow, as opposed to laminar flow, is characterised by an instantaneous and irregular deviation in the velocity profile relative to the mean velocity. This is visualised in Figure 2.1. Additionally, the repeatability of such instantaneous flow profiles is limited. Turbulent flows are extremely sensitive towards very small changes in the initial conditions, meaning that the instantaneous velocity profiles are not repetitive for similar cases. This is related to the non-linearity of turbulence systems [20]. For non-linear dynamic systems, it is known that initially infinitesimally close trajectories diverge when a small separation of the initial condition (δ) is present at a rate of $\delta \exp(\lambda t)$ [22]. In this, λ is the Lyapunov characteristic exponent, which is part of a spectrum of Lyapunov exponents. The maximum Lyapunov exponent determines the predictability of the trajectories, as a positive Lyapunov exponent will cause diverging trajectories and hence chaotic flow [23].

Thus, two main characteristics of turbulent flow can be listed [20].

- The velocity field fluctuates randomly in time and is highly disordered in space, exhibiting a wide range of length scales.
- The velocity field is unpredictable in the sense that a small change in the initial conditions will produce a large change to the subsequent motion.

Additionally, it must be noted that turbulent flow is inherently three-dimensional. Even in flows where only pressure or the mean velocity changes in one or two direction, the turbulent structures that arise can still be three-dimensional [21]. The final characteristic of turbulence that should be discussed is that turbulent flow is dissipative, meaning that energy is transferred from a large scale to a small scale, where it is dissipated into heat. In literature, this process is usually referred to as the energy cascade.

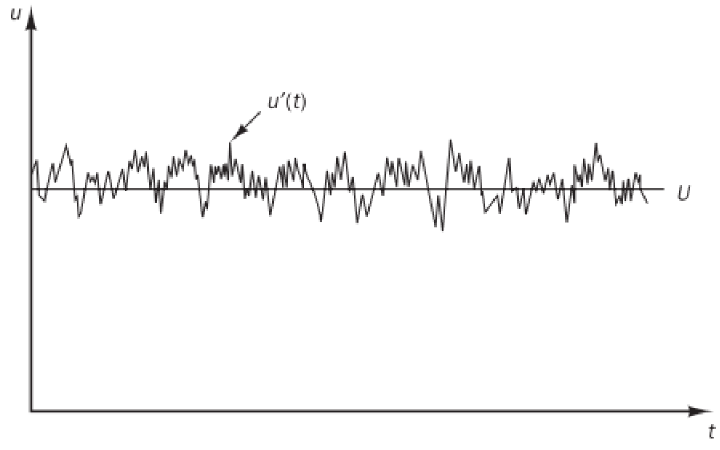


Figure 2.1: The characteristic random and irregularly fluctuating velocity profiles in turbulent flow. In this graph, U denotes the average velocity and $u'(t)$ denotes the instantaneous velocity. Reproduced from [21].

2.1.1 Energy cascade

Flow structures that are formed in turbulent media are vortices, which are usually referred to as eddies. These eddies can range in size over a broad scale, which is important for the energy dissipation. The largest eddies can undergo vortex stretching, which is the phenomenon in which the large eddies absorb energy from the mean flow and perform deformation work on the smaller eddies. Meanwhile, the vortices stretch in one direction, which causes an increment in radius in the other two directions to obey angular momentum conservation. This will continue until an unstable eddy radius is reached, after which eddy break-up will occur. The energy that is contained by these eddies as a function of the characteristic frequency is termed the energy spectrum of turbulence ($E(\kappa)$, with κ being the eddy wavenumber). This spectrum is visualised in Figure 2.2. From this, it can be seen that the smaller eddies (higher wavenumber) contain less turbulent energy. The total turbulence kinetic energy can be computed from the energy spectrum, which is represented in Equation 2.1 together with the definition of the turbulent kinetic energy. In here, $\overline{u_i'u_i'}$ denotes the ensemble averaged square of the velocity fluctuations.

$$k = \frac{1}{2}(\overline{u_i'u_i'}) = \int_0^\infty E(\kappa)d\kappa \tag{2.1}$$

From Figure 2.2, it can be clearly seen that three regimes are present. Two main length scales, the integral scale (L_0) and the Kolmogorov microscale (η), describe the transitions between these regimes. The integral scale is the largest eddy scale in a turbulent system which contains the most turbulent kinetic energy, meaning that the size is limited by the flow dimensions. The Kolmogorov microscale is the smallest eddy length scale, which is characterised by a local Reynolds number in the order of unity [25]. This means that the viscous stresses are sufficiently large to dissipate the turbulent kinetic energy into heat. The mathematical formulation for this length scale is given in Equation 2.2 [26]. The regime in between the integral scale and Kolmogorov microscale (which is termed the inertial range) was studied theoretically by Kolmogorov and an expression was deduced for the dependency of the spectral energy on the wavenumber, which is later referred to

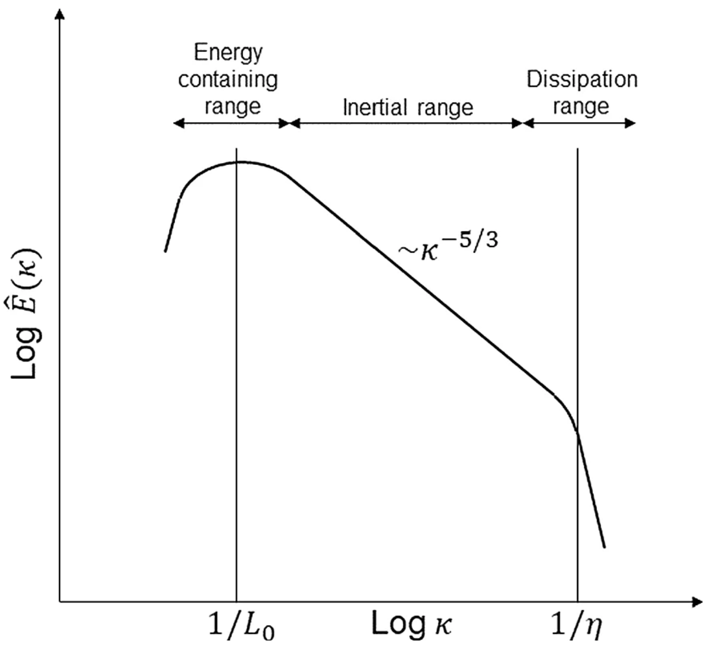


Figure 2.2: The spectral energy distribution of turbulence as part of the energy cascade. In this figure, L_0 denotes the integral scale and η denotes the Kolmogorov microscale. Reproduced from [24].

as "Kolmogorov's -5/3 law" [25, 26].

$$\eta = \left(\frac{\nu^3}{\varepsilon}\right)^{\frac{1}{4}} \tag{2.2}$$

Whereas the randomly fluctuating instantaneous velocity profiles are difficult to model due to the aforementioned reasons, the mean velocity and turbulence characteristics are possible to predict by means of models. The method to do so for complex flow geometries is the field of computational fluid dynamics, which will be discussed in Section 2.2.

2.2 Computational Fluid Dynamics

From the 1960s, the use of Computational Fluid Dynamics (CFD) in research and design in various industries was initiated and with the increasing computational resources available, CFD has become a vital tool in the design of industrial processes. CFD can be defined as the analysis of systems involving fluid flow, heat transfer and potential other associated phenomena such as chemical reactions by means of computer-based simulations [21], which revolve around numerically solving the Navier-Stokes equations.

2.2.1 Navier-Stokes equations

The Navier-Stokes equations can be viewed as mathematical equations expressing the conservation of momentum and mass for fluids, which determine the motion of the fluids. In Equation 2.3, the Navier-Stokes equation for a generalised fluid is given. In Equation 2.4, the continuity equation is depicted, which must be valid for incompressible flows.

$$\frac{\partial}{\partial t}(\rho \boldsymbol{u}) + \nabla \cdot (\rho \boldsymbol{u}\boldsymbol{u}) = -\nabla p + \nabla \cdot \boldsymbol{\tau} + \rho \mathbf{g} \tag{2.3}$$

$$\nabla \cdot \boldsymbol{u} = 0 \tag{2.4}$$

When Newton's law of viscosity (Equation 2.5) is applied, which states that the stress tensor is proportional to the gradient of the velocity with a proportionality constant being the dynamic viscosity (μ), Equation 2.6 can be derived.

$$\boldsymbol{\tau} = \mu \nabla \boldsymbol{u} \tag{2.5}$$

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot (\boldsymbol{u}\boldsymbol{u}) = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \boldsymbol{u} + \mathbf{g} \tag{2.6}$$

For complex systems, no analytical solution exists. Therefore, the equations have to be solved numerically by means of discretisation. Direct Numerical Simulation (DNS) is one method to solve these equations numerically. However, to simulate turbulent flow, DNS requires that the flow structures should be resolved to the Kolmogorov microscale. It is known that the spatial Kolmorogov microscale scales with the Reynolds number according to Equation 2.7, in which L_0 denotes the integral scale. Consequently, the number of grid cells in a domain can be determined to scale analogously with the Reynolds number. as shown in Equation 2.8. The number of time steps required to simulate a certain time domain can also be related to the Reynolds number, since the (maximum) time step should equal the interaction time at the Kolmogorov microscale in order to get a numerically stable solution. This is expressed in Equation 2.9. The computational time can be estimated to be proportional to the product of the number of grid cells on a domain cubed, since turbulence is inherently three-dimensional, and the number of time steps. This results in a scaling law of computational time with the Reynolds number cubed. Therefore, using DNS to solve turbulence problems with high Reynolds number is infeasible in many cases and different solution strategies should be used.

$$\eta \sim Re^{-\frac{3}{4}} L_0 \tag{2.7}$$

$$N_x \sim \frac{L_{domain}}{\Delta x} \sim \frac{L_{domain}}{L_0} Re^{\frac{3}{4}} \tag{2.8}$$

$$N_t \sim \frac{t_{sim}}{\Delta t} \sim \frac{t_{sim}}{\eta/u} \sim \frac{t_{sim}}{L_0/u} Re^{\frac{3}{4}} \tag{2.9}$$

$$t_{comp} \sim N_x^3 N_t \sim \left(\frac{t_{sim}}{L_0/u}\right)\left(\frac{L_{box}}{L_0}\right)^3 Re^3 \tag{2.10}$$

2.2.2 Reynolds-averaged Simulation

One of such different solution strategies is modelling turbulence by means of Reynolds-averaged simulation (RAS). This numerical technique consists of solving the Navier-Stokes equations for time-averaged variables. The methodology is started by performing a Reynolds decomposition on the velocity and pressure, which is represented in Equation 2.11 and 2.12. In this, the parameters are split into a time-averaged and fluctuating quantity. With the substitution of the decomposed pressure and velocity into the Navier-Stokes equations, the time-averaged Navier-Stokes equation can be derived after some simple algebraic modifications and the assumption of commutativity of the time averaging. This equation is depicted in Equation 2.13.

$$\boldsymbol{u} = \overline{\boldsymbol{u}} + \boldsymbol{u}' \tag{2.11}$$

$$p = \overline{p} + p' \tag{2.12}$$

$$\frac{\partial \overline{\boldsymbol{u}}}{\partial t} + \nabla \cdot (\overline{\boldsymbol{u}\boldsymbol{u}}) = -\frac{1}{\rho}\nabla \overline{p} + \nu \nabla^2 \overline{\boldsymbol{u}} + \nabla \cdot \overline{\boldsymbol{u}'\boldsymbol{u}'} + \boldsymbol{g} \tag{2.13}$$

Compared to the instantaneous Navier-Stokes equation (Equation 2.6), an additional term arises that represents the turbulent stresses ($\overline{\boldsymbol{u}'\boldsymbol{u}'}$), which are typically referred to as the Reynolds stresses. To close the system of equations, it is necessary to develop a turbulence model to predict the Reynolds stresses. Throughout the last decades, several turbulence models were established that can predict the Reynolds stresses for RAS simulations. The most common turbulence models are given in Table 2.1, ranked on basis of the computational costs of the model.

Number of additional transport equations	**Turbulence model**
0	Prandtl's mixing length model
2	$k-\varepsilon$ model $k-\omega$ model
7	Reynolds Stress model

Table 2.1: Turbulence closure models for RAS in the order of increasing numerical complexity.

A complete and detailed overview of all relevant RAS turbulence closure models is beyond the scope of this research. However, an attempt will be made to elaborate upon the working principle of these turbulence models. Most of the simple turbulence models, such as Prandtl's mixing length models and the $k-\varepsilon$ and $k-\omega$ models are based on the Boussinesq assumption, which states that the Reynolds stresses are proportional to the mean rates of deformation, analogous to the viscous stresses in Newton's law of viscosity. The Boussinesq assumption is given in Equation 2.14. In this equation, k is the turbulent kinetic energy per unit of mass and ν_t is the turbulent eddy viscosity. The factor δ_{ij} denotes the Kronecker delta, which has a value of 1 if $i = j$ and has a value of 0 if $i \neq j$. These relatively simple turbulence models revolve around accurate estimations of the turbulent eddy viscosity. To exemplify, Prandlt's mixing length model estimates the turbulent eddy viscosity on basis of the mean velocity gradient and the $k-\varepsilon$ model estimates the turbulent eddy viscosity by means of solving transport equations for the turbulent kinetic energy per unit of mass (k) and the rate of dissipation of turbulent kinetic energy per unit of mass (ε). Analogously, the $k-\omega$ model uses the turbulence frequency (ω) as a second variable for estimating the turbulent eddy viscosity.

$$-\overline{\boldsymbol{u_i}'\boldsymbol{u_j}'} = \nu_t \left(\frac{\partial \overline{u_i}}{\partial x_j} + \frac{\partial \overline{u_j}}{\partial x_i} \right) - \frac{2}{3} k \delta_{ij} \tag{2.14}$$

These simple turbulence models are incapable of dealing with anisotopic turbulence due to the fact that the underlying assumptions (mainly the Boussinesq approximation) assume that the Reynolds stresses are proportional to the mean rate of deformation of the main flow. Therefore, these models will fail to represent correct turbulence behaviour at conditions at which anisotropic turbulence will occur. Additionally, the presence of extra body forces may contribute to the breakdown of this model. To avoid the use of the Boussinesq assumption and to include complex effects, the Reynolds Stress model was developed. This model solves the transport equations for the Reynolds stresses directly, including the complex turbulent processes such as turbulent diffusion of the Reynolds stresses, production of turbulence, dissipation and inhomogeneity of the turbulence. With the introduction of these transport equations, the computational costs rise, but so does the possibility of capturing complex turbulent behaviour.

2.2.3 Large Eddy Simulation

Instead of averaging the Navier-Stokes equations temporally as in RAS, the Navier-Stokes equations can also be spatially averaged using a filtering function. This is the underlying principle of Large Eddy Simulation (LES). Using this approach, the Navier-Stokes equations can be solved up to some spatial resolution for the filtered velocity and pressure. As opposed to RAS, LES is also able to model a part of the turbulence characteristics. The extent to which the turbulence will be modelled is dependent on the selected filter size and it should be selected such that the largest eddies are resolved. Resolving these is important, as the large eddies contribute the most to the total turbulent kinetic energy (see Figure 2.2). The small-scale structures are not resolved and their effect on the mean flow is modelled. Hence, the computational costs are lower compared to DNS, while the turbulent nature is modelled more accurately than for RAS with a justifiable increase of computational costs. Since turbulent energy tends to dissipate from the large eddies towards the smaller eddies via the energy cascade, the approach of modelling the smaller eddies with closure models is justified. However, it must be noted that influence of the small-scale eddies on the large-scale eddies is not accounted for correctly in LES [27]. The uncertainty contained in the small-scale eddies can propagate to the large-scale eddies, leading to a mathematical ill-posed problem [27]. Even though this uncertainty in the small-scale eddies might lead to inaccurate flow behaviour, the overall statistical properties resemble the modelled situation in many cases [28].

Filtering and filtering functions

Multiple methods of filtering the Navier-Stokes equations exist. These approaches can be subdivided into explicit filtering and implicit filtering. In explicit filtering, a filter function is applied to the original function. The most used filter function in implementations of LES in CFD is the top-hat filter, which is depicted for a three-dimensional case in Equation 2.15. This filter effectively averages the original function such that fluctuations of scales much less than the cutoff width Δ are smoothened out. More advanced filtering functions are the Gaussian filter (Equation 2.16) and the spectral cutoff filter (Equation 2.17), but the use of these in finite volume implementations of LES remains limited [21]. Implicit filtering is filtering of the of the fluctuations by an implementation in the discretisation of the Navier-Stokes equations. This is similar to the explicit implementation by application of a top-hat filter. For this reason, the cutoff width is related to the grid cell sizes, as the filtering is limited in resolution by the grid cell size. Therefore, the cutoff width is mostly selected to be the cube root of the grid cell volume, as given in Equation 2.18.

$$G(\boldsymbol{x}, \boldsymbol{x'}, \Delta) = \begin{cases} 1/\Delta^3 & |\boldsymbol{x} - \boldsymbol{x'}| \geq \Delta/2 \\ 0 & |\boldsymbol{x} - \boldsymbol{x'}| > \Delta/2 \end{cases} \tag{2.15}$$

$$G(\boldsymbol{x}, \boldsymbol{x'}, \Delta) = \left(\frac{\gamma}{\pi\Delta^2}\right)^{\frac{3}{2}} \exp\left(-\gamma\frac{|\boldsymbol{x} - \boldsymbol{x'}|^2}{\Delta^2}\right) \tag{2.16}$$

$$G(\boldsymbol{x}, \boldsymbol{x'}, \Delta) = \prod_{i=1}^{3} \frac{\sin\left((x_i - x_i')/\Delta\right)}{(x_i - x_i')} \tag{2.17}$$

$$\Delta = \sqrt[3]{\Delta x \Delta y \Delta z} \tag{2.18}$$

Filtering the Navier-Stokes equations

When it is assumed that the filtering operations are commutative, the LES Navier-Stokes equations can be derived similarly to the derivation of the Reynolds-averaged Navier-Stokes equations. The LES Navier-Stokes equations are depicted in Equation 2.19. In this equation, the tilde denotes the filtered quantity.

$$\frac{\partial \widetilde{\boldsymbol{u}}}{\partial t} + \nabla \cdot (\widetilde{\boldsymbol{u}\boldsymbol{u}}) = -\frac{1}{\rho}\nabla \tilde{p} + \nu\nabla^2\widetilde{\boldsymbol{u}} + \boldsymbol{g} \tag{2.19}$$

However, this equation contains a term which requires information of the non-filtered velocity $((\widetilde{\boldsymbol{u}\boldsymbol{u}}))$, which is unknown. Therefore, a mathematical operation, as depicted in Equation 2.20, is performed to capture this using the introduction of the sub-grid-scale stress tensor $\boldsymbol{\tau}_{SGS}$. This can be substituted into the derived Navier-Stokes equations to yield the filtered Navier-Stokes equations as shown in Equation 2.21. Similar to the RAS case, this again yields an unclosed system, in which the sub-grid-scale stress tensor should be modelled using a closure model.

$$\nabla \cdot (\widetilde{\boldsymbol{u}\boldsymbol{u}}) = \nabla \cdot (\widetilde{\boldsymbol{u}}\widetilde{\boldsymbol{u}}) + (\nabla \cdot (\widetilde{\boldsymbol{u}\boldsymbol{u}}) - \nabla \cdot (\widetilde{\boldsymbol{u}}\widetilde{\boldsymbol{u}})) = \nabla \cdot (\widetilde{\boldsymbol{u}}\widetilde{\boldsymbol{u}}) + \nabla \cdot \boldsymbol{\tau}_{SGS} \tag{2.20}$$

$$\frac{\partial \widetilde{\boldsymbol{u}}}{\partial t} + \nabla \cdot (\widetilde{\boldsymbol{u}}\widetilde{\boldsymbol{u}}) = -\frac{1}{\rho}\nabla \tilde{p} - \nabla \cdot \boldsymbol{\tau}_{SGS} + \nu\nabla^2\widetilde{\boldsymbol{u}} + \boldsymbol{g} \tag{2.21}$$

In order to better understand the sub-grid-scale stress tensor, the velocity can be decomposed into a filtered quantity and a fluctuating quantity, as performed in Equation 2.22. When this is substituted into the definition of $\boldsymbol{\tau}_{SGS}$, three contributions can be derived, which are the Leonard stresses, cross stresses and the Reynolds stresses for LES. The Leonard stresses are stresses that are related to effects on the resolved scale. The cross stresses are related to interactions between the resolved flow and the sub-grid-scale structures. Finally, the LES Reynolds stresses are caused by momentum transfer of the sub-grid-scale eddies with the resolved flow, similar to the Reynolds stresses as discussed for RAS. These stresses must be modelled using closure models in order to solve the system of equations. Despite the different nature of the stresses, sub-grid-scale turbulence models estimate the sub-grid-scale stress tensor by means of a single model [27]. The most common LES turbulence models are briefly discussed hereafter.

$$\boldsymbol{u} = \widetilde{\boldsymbol{u}} + \boldsymbol{u'} \tag{2.22}$$

$$\boldsymbol{\tau}_{SGS} = \widetilde{\boldsymbol{u}\boldsymbol{u}} - \widetilde{\boldsymbol{u}}\widetilde{\boldsymbol{u}} = \left(\widetilde{\widetilde{\boldsymbol{u}}\widetilde{\boldsymbol{u}}} - \widetilde{\boldsymbol{u}}\widetilde{\boldsymbol{u}}\right) + \widetilde{\widetilde{\boldsymbol{u}}\boldsymbol{u'}} + \widetilde{\boldsymbol{u'}\boldsymbol{u'}} \tag{2.23}$$

$$\boldsymbol{\tau}_{SGS} = \boldsymbol{L} + \boldsymbol{C} + \boldsymbol{R} \tag{2.24}$$

- Leonard stresses: $\boldsymbol{L} = \left(\widetilde{\widetilde{\boldsymbol{u}}\widetilde{\boldsymbol{u}}} - \widetilde{\boldsymbol{u}}\widetilde{\boldsymbol{u}}\right)$
- Cross stresses: $\boldsymbol{C} = \widetilde{\widetilde{\boldsymbol{u}}\boldsymbol{u'}}$
- Reynolds stresses for LES: $\boldsymbol{R} = \widetilde{\boldsymbol{u'}\boldsymbol{u'}}$

Sub-grid-scale stress models

The most frequently used sub-grid-scale stress models are based on eddy viscosity models, similar to the turbulence closure models for RAS. For this, the Boussinesq assumption is applied, as depicted in Equation 2.25 with the deformation tensor of the filtered field as defined in Equation 2.26. Since this assumption now only concerns the sub-grid-scale eddies, the problem with anisotropy is mitigated, as smaller eddies are generally more isotropic [25]. Then, turbulence closure models can be used to estimate the kinematic turbulent eddy viscosity ν_t.

$$\tau_{ij}^{SGS} = 2\nu_t \widetilde{S_{ij}} + \frac{1}{3}\delta_{ij}\tau_{ll}^{SGS} \tag{2.25}$$

$$\widetilde{S_{ij}} = \frac{1}{2}\left(\frac{\partial \widetilde{u_i}}{\partial x_j} + \frac{\partial \widetilde{u_j}}{\partial x_i}\right) \tag{2.26}$$

Smagorinsky-Lilly

The most common turbulence model in LES is the Smagorinksy-Lily model, which originates from 1963 [29]. In this model, an eddy viscosity was introduced that should represent the three-dimensional turbulence in the sub-grid scales. It was established on the foundations of Prandlt's mixing length model, with an eddy viscosity that is proportional to a characteristic length scale and velocity scale. This length scale was selected to be the cutoff width and as in Prandlt's model, the velocity scale is modelled by the product of this length scale and the average strain rate of the resolved flow. The concluding equation for the kinematic turbulent eddy viscosity is then given in Equation 2.27. In this equation, the constant C_s is still present. Lily adapted the Smagorinsky model by a theoretical study for appropriate values of this constant [30, 31]. By assuming that wavenumber of the eddies associated to the cutoff width lie within the Kolmogorov inertial range, meaning that the turbulent kinetic energy scales with $k^{-5/3}$, a value of 0.18 can be found. This is found by adjusting the value of this constant such, that the ensemble averaged sub-grid turbulent kinetic energy dissipation should be identical to the turbulent energy dissipation rate ε. However, still many different values of this constant are used in literature, as it seems that different cases require different values of C_s in order to produce accurate results which lack overdamping [21]. Additionally, the Smagorinsky model is too dissipative close to walls when compared with theoretical profiles close to the wall [27]. This indicates that the application of this model is not universal and a more sophisticated approach might be necessary.

$$\nu_t = (C_s \Delta)^2 \sqrt{2\widetilde{S_{ij}}\widetilde{S_{ij}}} \qquad \text{with } C_s \approx 0.18 \tag{2.27}$$

Wall-adapting local eddy viscosity

A turbulent viscosity model that was developed in order to mitigate the problems associated with the classical Smagorinsky model is the wall-adapting local eddy-viscosity (WALE) model. It was developed by Nicoud and Ducros in 1999 [32]. A main characteristic of the model is that it is both a function of the rate of strain as well as the rotation rate, as these both contribute to turbulent structures. Additionally, the model should behave such that no additional wall damping is necessary and the no-slip boundary condition is naturally reproduced with the proper wall scaling. The corresponding model that complies with these requirements is depicted in Equations 2.28 - 2.30. In this, the constant C_w is a constant of which a value of $0.55 \leq C_w \leq 0.60$ was found by equalizing the ensemble-averaged sub-grid kinetic energy dissipation rate to this of the classical Smagorinsky model [32]. With this constant, the proper asymptotic behaviour of the eddy viscosity close to solid walls was found, making the WALE model more appropriate for wall-bounded flows than the classical Smagorinsky model.

$$\nu_t = (C_w \Delta)^2 \frac{\left(S_{ij}^d S_{ij}^d\right)^{3/2}}{\left(\widetilde{S_{ij}}\widetilde{S_{ij}}\right)^{5/2} + \left(S_{ij}^d S_{ij}^d\right)^{5/4}} \tag{2.28}$$

$$S_{ij}^d = \frac{1}{2}\left(\widetilde{g_{ij}}^2 + \widetilde{g_{ji}}^2\right) - \frac{1}{3}\delta_{ij}\widetilde{g_{kk}}^2 \tag{2.29}$$

$$\widetilde{g_{ij}} = \frac{\partial \widetilde{u_i}}{\partial x_j} \tag{2.30}$$

2.2.4 Extensions towards multiphase CFD

For the purpose of simulating bacterial cells in flow, a multiphase CFD approach is required. In literature, the number number of studies to simulate bacteria in flow using CFD remains limited, yet the deformation of biological cells was widely researched for cardiovascular systems. Especially, the red blood cells are known to be able to deform in shear flows and are likely to aggregate in certain cases [33]. Many simulation strategies were developed to study the effect of shear rates, flow rate or flow morphology on blood flow [34–37]. Amongst these, Euler-Euler and Euler-Langrange models are present. In the sections below, the mathematical backgrounds of these types of models are given.

Euler-Euler models

In the Euler-Euler approach, the two phases are treated as interpenetrating continua [10]. This means that solid phases can be considered as a pseudo-fluid [38]. The phases are identified by means of a volume fraction. This volume fraction (α) for phase k can be solved using the conservation equation as in Equation 2.31. The momentum conservation equation is given in Equation 2.32 [39]. In this equation, the two interfacial forces are the drag force $\boldsymbol{F}_{D,k}$ and the surface tension force $\boldsymbol{F}_{s,k}$.

$$\frac{\partial \alpha_k}{\partial t} + \boldsymbol{u}_k \cdot \nabla \alpha_k = 0 \tag{2.31}$$

$$\frac{\partial \left(\rho_k \alpha_k \boldsymbol{u}_k\right)}{\partial t} + \nabla \cdot \left(\rho_k \alpha_k \boldsymbol{u}_k \boldsymbol{u}_k\right) = -\alpha_k \nabla p + \nabla \cdot \left(\mu_k \alpha_k \nabla \boldsymbol{u}_k\right) + \rho_k \alpha_k \boldsymbol{g} + \boldsymbol{F}_{D,k} + \boldsymbol{F}_{s,k} \tag{2.32}$$

The drag force can be described as in Equation 2.33, where subscripts c and d denote the continuous and dispersed phase. For the drag coefficient C_D, several correlations can be used, such as the Schiller and Neumann correlation [39]. The force related to the surface tension can be expressed using the continuum surface force (CSF) model [40].

$$\boldsymbol{F}_{D,k} = \frac{3}{4} \rho_c \alpha_c \alpha_d C_D \frac{\left|\boldsymbol{u}_d - \boldsymbol{u}_c\right| \left(\boldsymbol{u}_d - \boldsymbol{u}_c\right)}{d_d} \tag{2.33}$$

In order to implement such Euler-Euler model, a method to capture the interface between the two phases must be established. In simulations performed on red blood cells, the Volume of Fluid (VoF) method showed to accurately capture this interface [33–35, 37]. This can be done by implementing a colour function $F = f(x, y, z, t)$, which has a value between zero and unity [41]. It indicates the fractional amount of fluid present at a certain position at a certain time [42]. This fraction can again be related to the volume fraction. In order to reduce numerical smearing, a compressive VoF technique was introduced, which yields sharp interfaces of similar quality to interfaces constructed from geometric reconstruction schemes using piecewise linear interface calculation [43]. In this compressive approach, an artificial interface-compression velocity $\boldsymbol{u}_c$ is introduced, which is defined such that the gradient of the volume fraction steepens near the interface. Using this strategy, the conservation equation for the volume fraction as defined in Equation 2.31 can be redefined to Equation 2.34. The term $\alpha_k \left(1 - \alpha_k\right)$ ensures that only near the interface, meaning when volume fractions are in between 0 and 1, the interface compression is active [43].

$$\frac{\partial \alpha_k}{\partial t} + \boldsymbol{u}_k \cdot \nabla \alpha_k + \nabla \cdot \left(\boldsymbol{u}_c \alpha_k \left(1 - \alpha_k\right)\right) = 0 \tag{2.34}$$

The interface compression velocity is given by Equation 2.35. In this, C_α is the compression coefficient, which determines the magnitude of the interface compression velocity. In case of a compression coefficient of 0, no interface compression takes place and for a compression coefficient of 1, sharp interface capturing is applied.

$$\boldsymbol{u}_c = \min(C_\alpha |\boldsymbol{u}|, \max(|\boldsymbol{u}|)) \frac{\nabla \alpha}{|\nabla \alpha|} \tag{2.35}$$

A main benefit of the VoF model using interface compression is that it is much easier to implement and performs faster compared to other surface capturing techniques [42, 43].

Additionally, this method is mass conservative, in contrast to methods like the level set method [42]. A drawback of this model is that merging of interfaces can occur automatically.

Euler-Lagrange models

In the Euler-Lagrange method, the fluid phase is still modelled as a continuum using the Navier-Stokes equations, but the dispersed phase is solved by tracking the particles individually [38]. Using the calculated flow field, forces on the particles can be computed and the motion can be obtained by Newton's second law and the equations for rotation [10]. Interparticle collision can be modelled using a contact model. Two main contact models are the hard-sphere model and the soft-sphere model [44]. In the soft-sphere approach, the particles can overlap and deform using a linear spring/dashpot model, from which a repulsive force arises [44]. In the hard-sphere approach, the particles cannot overlap and the interaction forces are impulsive. The total impulse acting on a particle can be computed using the coefficients of restitution and coefficient of friction. Using this impulse, the post-collision motion can be computed. Since the fluid field is affected by the particles and vice-versa, adequate coupling between the Eulerian and Lagrangian solvers is necessary. This can be done in different ways, such as one-way (fluid-particle interactions), two-way (fluid-particle and particle-fluid interactions) and four-way coupling (fluid-particle, particle-fluid, particle-particle and particle-wall interaction) [45]. Due to this, the complexity and computational time is much greater compared to Eulerian-Eulerian models. Elghobashi studied the regimes at which each type of fluid-particle coupling becomes important and these are depicted in Figure 2.3. On basis of the volume fraction of particles (α_p) and either the ratio between the particle response time and the Kolmogorov time (τ_p/τ_K) or the ratio between the particle response time and the eddy turnover time (τ_p/τ_e), the type of coupling required can be determined. At low volume fractions of particles, the momentum of the particles has negligible effects on the flow and turbulence and one-way coupling (regime 1) is sufficient. At larger volume fractions, the particles will affect the turbulence and two-way coupling should be used. This is divided into regime 2 and regime 3 on basis of the nature of interaction with the turbulence. At low values of the particle response time, the particles will cause an increased dissipation rate (regime 3) and at higher values of τ_p, the particles will enhance the production of turbulent kinetic energy (regime 2). When the volume fraction of the particles is increased further, a more dense suspension of particles is obtained and particle-particle interaction will become important, making it important to use four-way coupling (regime 4).

Deformation

In order to accurately simulate the deformation of red blood cells in flow, different deformation models were included in CFD models. Ju et al. [48] reviewed the numerical methods to describe this cell deformation. One of these deformation models is the shell-based deformation model, in which the membrane is modelled as an infinitesimally thin shell that can deform, leading to interfacial tension. This interfacial membrane tension can be decomposed into an in-plane tension and a transverse shear tension. These separate tensions can then be related to constitutive equations, such as a neo-Hookean model. Another method to model the deformation is by means of spring-based models. These models consist of a network of springs and dampers that can represent the viscoelasticity in cells. In this method, the cell is discretised by a set of points which are interconnected

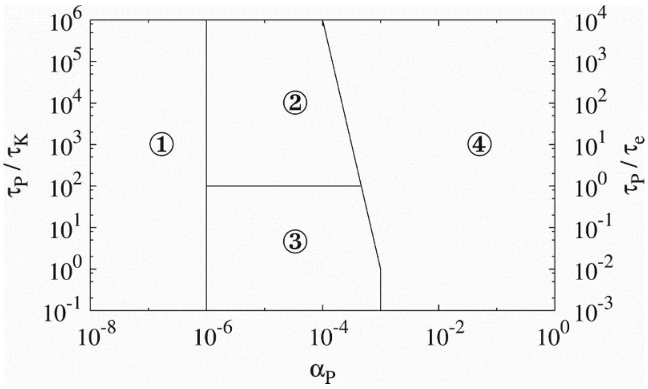


Figure 2.3: Classification of the Euler-Lagrange coupling strategies for particle-laden flows according to Elghobashi [46]. Reproduced from [47]

by springs. This method was applied to Stokes flow for 2D cases [49]. Implementations of these models towards 3D cases and highly turbulent fields have yet to be investigated, as the flow is restricted to Stokes flow [33]. Another method of implementing the deformation in cells is by means of describing the cell as a viscous droplet in a full Eulerian simulation [37]. The viscoelasticity can then be implemented via constitutive equations that describe the effective viscosity on basis of the shear rate amongst other parameters [50]. The benefit of this technique is that it can be easily implemented in standard multiphase flow solvers combined with VoF and it enables 3D simulation due to lower (numerical) complexity [33].

2.3 Hydrodynamics in the rotor-stator spinning disc reactor

A pioneering work in the hydrodynamics of enclosed rotating discs without superimposed throughflow was performed by Daily and Nece [51]. On basis of two main parameters, the rotational Reynolds number (Re_ω as defined in Equation 2.36) and the ratio between the axial clearance of rotor and stator and disc radius (G, as defined in Equation 2.36), four hydrodynamic regimes were encountered. The classification of the flow regimes was denoted as follows.

- Regime I: Laminar flow, merged boundary layers
- Regime II: Laminar flow, separated boundary layers
- Regime III: Turbulent flow, merged boundary layers
- Regime IV: Turbulent flow, separated boundary layers

Later, an overview was made by Launder et al. [52] that maps the hydrodynamic regimes on basis of the rotational Reynolds number Re_ω and the dimensionless interdisc spacing G. This overview is given in Figure 2.4.

$$Re_\omega = \frac{\omega_{rot} r_d^2}{\nu} \quad \text{with } G = \frac{s}{r_d} \tag{2.36}$$

The boundary layers as mentioned in the work by Daily and Nece [51] were theoretically studied by Von Kármán and Bödewadt. The work by Von Kármán consists of a study

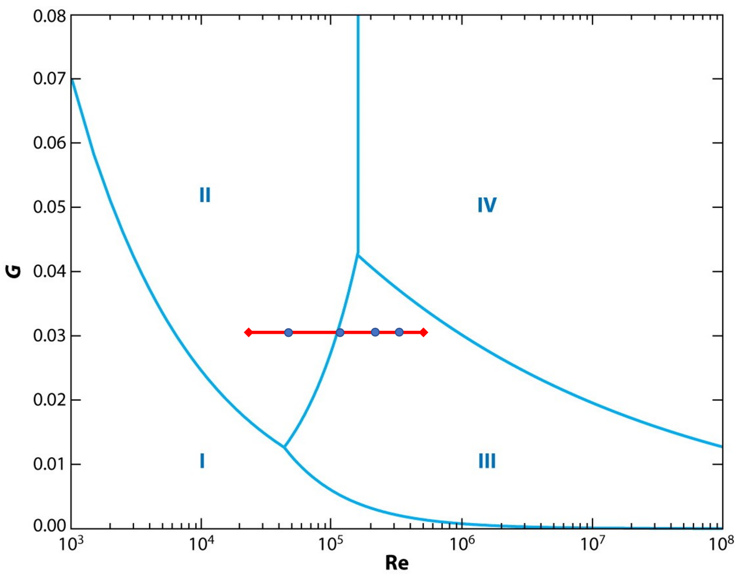


Figure 2.4: Map of the four flow regimes proposed by Daily and Nece [51]. Merged boundary layers: I (laminar) and III (turbulent). Unmerged boundary layers: II (laminar) and IV (turbulent). Reproduced from [52]. The red line denotes the range of operation for experiments, whereas the points denote the simulations.

of a rotatating disc in quiescent medium [53]. The boundary layer near the rotating disc that is formed in this process, named after Von Kármán himself, can be described by a net radial velocity in the positive direction and an axial velocity in the direction of the disc. Additionally, the velocity in the azimuthal direction is related to the linear velocity of the rotating disc.

Bödewadt studied the reversed case, in which a rotating fluid interacts with a stationary disc [54]. The boundary layer formed in this process is inversely related to the Von Kármán boundary layer, with a net radial velocity in the negative direction and an axial velocity away from the disc.

In the descriptions of the hydrodynamics in rotor-stator fluid flow, the velocity profiles can be viewed as a combination of the Von Kárman boundary layer and the Bödewadt boundary layer. With high gap ratio values, the boundary layers are separated, resulting in an inviscid core between the boundary layers. This inviscid core is characterised by having negligible viscous forces, which is related to the absence of shear stresses, rather than a negligible viscosity. A type of flow that can be observed in the separated boundary layer regime is Stewartson flow. In this flow type, the Bödewadt boundary layer is absent and only the boundary layer near the rotor is formed. The bulk of the fluid is at rest. A schematic overview of the Stewartson flow is given in Figure 2.5a. Another type of flow that can be observed in the separated boundary layer regime is Batchelor flow (Figure 2.5b). In this flow type, the Bödewadt boundary layer is formed at the stator and the inviscid core is rotating. The boundary conditions determine whether the Stewartson or Batchelor flow structures will be observed [18]. In case of a fully enclosed rotating disc, the flow structures that will arise are like Batchelor flow, whereas a superimposed external throughflow will cause Stewartson flow.

The importance of throughflow can be quantified using the flow rate coefficient (C_w), which is defined in Equation 2.37. In this definition, a positive volumetric flow rate (and hence a positive flow rate coefficient) denotes a centrifugal throughflow. In many cases,

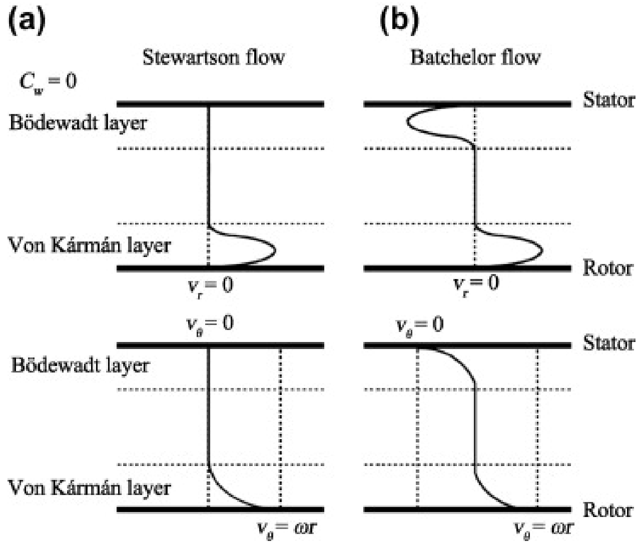


Figure 2.5: Schematic overview of the velocity profiles in the tangential and radial direction for Stewartson flow (a) and Batchelor flow (b). Reproduced from [18].

the external throughflow rate is negligible in comparison with the recirculation rate in the Von Kármán boundary layers [55]. However, the external throughflow can affect the flow structures significantly [56] and it further complicates the hydrodynamics in the rs-SDR.

$$C_W = \frac{\phi_V}{\nu r_d} \tag{2.37}$$

With external centrifugal throughflow, a Stewartson-like flow structure is observed at low radial positions [56, 57]. At these low radial positions, a Von Kárman boundary layer is formed near the rotor and the tangential velocity decreases to zero in the inviscid core that is formed. The radial velocity is positive for all axial positions. This flow regime can be described as a thoughflow governed regime. A schematic overview of this regime is given in Figure 2.6a.

At high radial positions, the flow structure becomes rotation governed and the flow structures will behave as a torsional Couette flow or a Batchelor flow [56–58]. The gap ratio will determine in this case whether the merged boundary layer hydrodynamics as in a torsional Couette flow will be dominant, or whether the flow will resemble the separated boundary layer flow structures as in Batchelor flow. With low gap ratio values, the boundary layers merge, which results in an absence of the inviscid core and a continuously changing tangential and radial velocity profile along the axial coordinate. In Figure 2.6b, the rotation governed flow structures in case of a torsional Couette flow is depicted.

So clearly, there are radial positions at which the transition from the throughflow governed regime towards the rotation governed flow is present. The radial position (non-dimensionalised with the disc radius) at which this transition happens was studied theoretically by Phadke et al. [59] and was found to be a function of the flow rate coefficient and the rotational Reynolds number. This dependence is depicted in Equation 2.38. Additionally, a constant (c) was introduced, of which the theoretical value (0.219) was confirmed by CFD Reynolds stress model simulations [58]. It must be noted that this result is only valid for the transition in the Batchelor regime. For the case of torsional Couette flow, no literature is available on predicting the radial position at which the transition occurs.

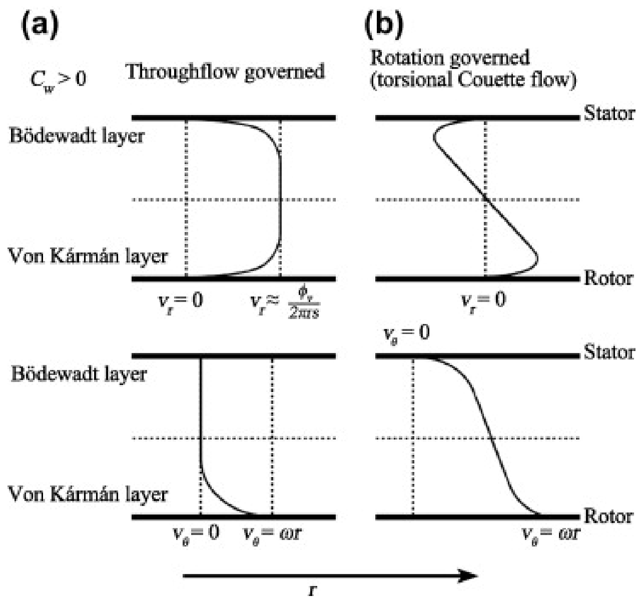


Figure 2.6: Schematic overview of the velocity profiles in the tangential and radial direction for throughflow governed (a) and rotation governed (b) flow regimes for a superimposed external centrifugal throughflow. Reproduced from [18].

Also, the gap ratio seems to have no effect on the radial position at which the transition occurs from the throughflow governed to the rotation governed region. For the Batchelor regime, literature research confirms that the gap ratio does not seem to be a relevant parameter [57, 60], yet the effect of the gap ratio in the torsional Couette flow is not adequately accounted for [18].

$$r^*_{trans} = \left(\frac{1}{c}\frac{C_w}{Re_\omega^{\frac{4}{5}}}\right)^{\frac{5}{13}} \tag{2.38}$$

The macromixing behaviour of fluid flow in the rs-SDR can be described by a combination of plug flow volume and n ideally stirred tanks-in-series [19], and hence the residence time distribution curve can be described by Equation 2.39. In here, τ_{pf} denotes the residence time of the plug flow governed region, τ_m denotes the mean residence time and n denotes the number of tanks-in-series.

$$E_{mod}(t) = \begin{cases} 0 & t < \tau_{pf} \\ \frac{t^{n-1}}{(n-1)!\left(\frac{\tau_m - \tau_{pf}}{n}\right)^n} \exp\left(-\frac{t}{\left(\frac{\tau_m - \tau_{pf}}{n}\right)}\right) & t \geq \tau_{pf} \end{cases} \tag{2.39}$$

As a final remark, it should be noted that the hydrodynamics discussed in this section are only applicable to Newtonian fluids. To take the effect of non-Newtonian fluids into consideration, numerical simulations in the field of computational fluid dynamics (CFD) are most likely required to tackle the complexity.

2.4 Microbial fermentation of syngas by *Clostridium autoethanogenum*

In 1993, the bacterial species *Clostridium autoethanogenum* (*C. auto)* was isolated from rabbit feces [61]. The culture consists of gram-positive, motile bacteria which can metabolise C1 compounds such as CO, H_2 and CO_2 or mixtures thereof with ethanol and acetate as end-products [61, 62]. In the next section, an overview of the bacterial properties is provided.

2.4.1 Bacterial properties

Using microscopy techniques, the size and shape of the bacteria were studied by Abrini et al. [61]. It was found that the micro-organisms appereared as rod-type bacteria with a considerable distribution in size. Depending on the culture age and feed source, bacterial lengths were found ranging from 2.1 to 9.1 μm. In old cultures, large filamentous cells up to 42.5 μm were observed. The distribution in the bacterial cell diameter was found to be less broad, with values ranging from 0.5 to 0.6 μm. An average bacterial cell diameter of 0.5 μm and length of 3.2 μm was found. The density of bacteria was studied by Bratbak et al. [63]. The buoyant density of the bacterial cells studied was found to range between 1.09 and 1.13 g cm^{-3}. Since the bacterial species in this study mimic *Clostridium autoethanogenum* in terms of morphology and chemically (gram-positive), the density of *Clostridium autoethanogenum* can be assumed to be within this range.

Using contact angle measurements of droplets of solvents on a solid surface of filtrated cells of *Clostridium carboxidivorans*, the bacterial surface tension was investigated [64]. The surface tension was measured to be 56 mJ m^{-2}, corresponding with a contact angle for water of 33°. Since the bacterial specie *Clostridium carboxidivorans* closely resembles *Clostridium autoethanogenum* in terms of chemical cell wall composition and morphology, it can be assumed that *Clostridium autoethanogenum* will also have this intermediately hydrophobic character.

As *Clostridium autoethanogenum* is gram-positive [61], it can be concluded that the principal component of the cell wall is peptidoglycan [65]. The mechanical properties of cell walls consisting of peptidoglycan was studied by Mendelson et al. [66]. Using bacterial thread made from peptidoglycan, stress-strain experiments were performed to study the viscoelastic properties of bacterial cell walls. It must be noted that the properties measured are actually those of the cell wall, because its multifilament construction might suggest that the mechanical properties observed do not apply to a single bacterial cell. However, due to the absence of slipping between filaments as measured in the stress-strain curves, the measured properties represent the bacterial cell wall [67].

It was found that the yield stress is highly dependent on the relative humidity. When wetted, the bacterial cell wall has a tensile strength of 3 MPa [67]. The stress-strain curves found behave much like other viscoelastic polymers, which enables fitting a viscosity to the stress-strain curves. It was found that a viscous material with a viscosity of 20 MPa$\cdot$h represents the experimentally obtained strain rates [68].

Chapter 3 Methods

3.1 Simulation of a rotor-stator cavity

The computational fluid dynamics simulation of flow in a rotor-stator cavity, and all subsequent CFD simulations, were performed in OpenFOAM. OpenFOAM is an acronym for Open Source Field Operation and Manipulation, which highlights the main aspects of the software [69]. It consists of a C++ library containing applications that fall into the categories of solvers and utilities [70]. To these, custom made applications can be added. These solvers are at the heart of the OpenFOAM toolbox. Additionally, pre-processing and post-processing utilities are presents, that can be used to initialise the problem on a mesh and visualise or analyse the results, respectively.

Numerical implementation

CFD analyses in OpenFOAM are based on the finite volume method, in which the computational domain is discretised into volume elements at which the equations can be solved [69]. Therefore, numerical schemes should be implemented. OpenFOAM makes predominantly use of collocated variable arrangements [70]. Thus, interpolation schemes are necessary to transform cell-based quantities to cell faces and convert volume integrals to surface integrals [70]. The interpolation schemes that were used for the simulation of the rotor-stator cavity are depicted in Table 3.1. Generally, the higher-order cubic interpolation scheme was used, since this scheme shows higher than second order convergence rates in some cases and yields better accuracy with lower numerical diffusion compared to standard OpenFOAM central differencing schemes [71]. Additionally, it captures the turbulent decay in terms of energy dissipation better, although it underestimates the turbulent energy dissipation rate [71]. For both the laplacian as well as the surface-normal gradient interpolation schemes, non-orthogonal correction was added. This is especially important for this case, since cylindrical geometries lead inevitably to non-orthogonal meshes [72]. However, the non-orthogonality of the mesh should still be minimised, as the correction steps for highly non-orthogonal meshes can lead to numerical instabilities [70]. For the divergence scheme, van Leer interpolation was used, which is a total variation diminishing (TVD) scheme [73]. This scheme ensures that unphysical spatial oscillations that can arise in convection-dominated flows with central differencing discretisation are suppressed, while keeping reasonable accuracy [74].

Next to the spatial discretisation, tempororal discretisation is necessary. The discretisation method that was selected in this simulation is the Crank-Nicolson method. The Crank-Nicolson scheme is second-order and implicit in time. This scheme can be implemented in OpenFOAM as a blended scheme, being a Crank-Nicolson scheme blended with the backward Euler scheme for stabilisation and limiting of numerical oscillations. A coefficient

Mathematical term	Interpolation scheme
Gradient	Gauss cubic
Divergence	Gauss van Leer
Laplacian	Gauss cubic corrected
Interpolation	Cubic
Surface-normal gradient	Cubic corrected

Table 3.1: Interpolation schemes that were used in the rotor-stator cavity simulations.

was introduced that determines the relative contribution of the Crank-Nicolson scheme and the backward Euler scheme, of which a value of 0 corresponds to a full backward Euler discretisation and a value of 1 corresponds to a full Crank-Nicolson discretisation. A value of 0.9 was selected to increase the accuracy as the Crank-Nicolson scheme is second order, but blend in the stabilizing effect of the backward Euler scheme [21].

Next to the numerical schemes, also solvers should be selected to solve the discretised system of algebraic equations. For the pressure field, the *GAMG* solver was selected and for the additional fields, the solver *smoothSolver* was used. This selection is based on the default solvers in tutorials of OpenFOAM [70]. The *GAMG* solver is a geometric-algebraic multigrid solver, which solves a set of equations in an efficient way by solving the problem on a coarser grid, which is then used to solve for the fine problem case [21]. In such algorithms, preconditioners and smoothers should be used to enhance the rate of convergence by smoothening the residuals. For *GAMG*, *GaussSeidel* and *FDIC* were selected to be the smoother and preconditioner, respectively. The tolerances for the residuals in pressure were set to $1 \cdot 10^{-6}$ and for the additional quantities to $1 \cdot 10^{-5}$.

When solving the equations for pressure and velocity, a coupling between these quantities is observed. In case of a compressible flow, an equation of state can be used to solve for the pressure. However, for incompressible cases a different solution strategy should be used. The velocity and pressure should be solved such that the pressure field yields via the momentum equations a velocity field which obeys the continuity equation. Algorithms that perform such solution strategy are iterative solvers, of which PISO and PIMPLE are the main algorithms [21]. In PISO, the velocity is computed in a first predictor step with an intermediate pressure field, after which the pressure field is corrected using a Poisson equation such that continuity is satisfied. This procedure is performed iteratively until convergence. PIMPLE adds an additional outer loop to this scheme. In each outer loop, the momentum predictor is solved once more with the converged pressure and velocity from the inner loop. Also, the PIMPLE algorithm adds the flexibility that the turbulence model is evaluated in each outer loop. Since the flow that is solved is highly turbulent, the PIMPLE algorithm was selected, with one inner corrector and five outer correctors at which the turbulence model is evaluated.

The WALE turbulence model for LES was selected for this case. Despite the fact that the Smagorinksy can be considered as a standard LES sub-grid-scale turbulence model, the drawbacks as described in Section 2.2.3 make the use of this turbulence model undesired. The main benefits of the use of WALE were discussed previously and one of these was that the rotational aspects of the turbulent flow are captured more adequately with the WALE turbulence model [32]. Especially this aspect of WALE makes the implementation of this turbulence model in rotor-stator flows suitable. The cutoff width was selected to be the cube root of the grid cell volume.

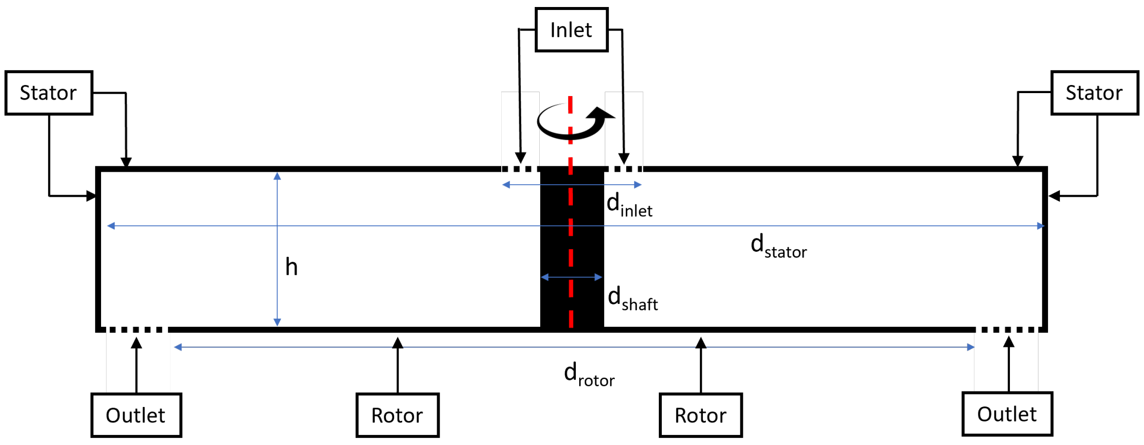


Figure 3.1: Schematic overview (not to scale) of a cross-section of the geometry used in the rs-SDR cavity simulations.

Computational mesh

The mesh that resembles the geometry of the rotor-stator cavity was constructed using the utility *blockMesh*. This utility constructs a mesh with hexahedral cells from certain blocks. The mesh was constructed from four blocks that correspond to a quarter of the annular region between rotor and stator. The circular boundaries near the edge and the shaft are made by arcs. These four blocks are combined to form the mesh. Using *topoSet*, the inlet and outlet regions on the mesh were defined. This results in a mesh which is schematically depicted in Figure 3.1. The dimensions are listed in Table 3.2.

Property	**Size [m]**
d_{inlet}	0.11
d_{shaft}	0.076
d_{rotor}	0.5
d_{stator}	0.56
h	0.009

Table 3.2: Dimensions of the geometry studied in the rs-SDR cavity simulation.

To resolve the boundary layer, a variable grid size in the wall-normal direction was used with refinement close to the walls. In the wall-normal direction, an expansion coefficient of 10 was used such that each cell closer to the wall has a size which is 10 times smaller. In the radial direction, the grid size was varied with an expansion coefficient of 5, resulting in a smaller grid size near the rim of the rotor and the shaft. In the tangential direction, no grid refinement was applied. In order to accurately resolve the boundary layer, multiple grid cells should be in the viscous sublayer. Therefore, the value of the dimensionless wall-normal coordinate y^+ should be below 1 [75, 76]. The definition of y^+ is given in Equation 3.1. In each simulation, the utility *yPlus* was used to check this requirement. In all simulations, the value of y^+ was well below 1. The grid size that was found to be adequate for resolving the boundary layer and keeping computational time reasonable was found to be N_θ x N_r x N_z = 300 x 500 x 140. This resulted in a total number of grid cells of 21 million. For this grid, the utility *checkMesh* was used to compute parameters that indicate the mesh quality. The maximum non-orthogonality was computed to be less than 1° and the maximum aspect ratio was approximately 200.

$$y^+ = \frac{y v_\tau}{\nu} \tag{3.1}$$

Initialisation and boundary conditions

The initial condition for the pressure was selected to be a uniform pressure of 0, with a *zeroGradient* boundary condition at each wall and the inlet and outlet. For the turbulent viscosity, an initial value of the kinematic viscosity of water was given. The boundary conditions on the walls were selected to be *fixedValue* to a value of 0. For the inlet and outlet, the turbulent viscosity is calculated by the turbulence model. The initial velocity field is initialised such that the velocity at radial positions lower than the rotor radius equals 40 percent of the rotational velocity of the rotor at that radial position. This value of 0.4 was selected after literature review on similar systems [56, 57], yet this value does not affect the steady-state solution as its purpose is merely to decrease the time at which a steady state is reached. At radial positions larger than the rotor radius, the velocity is initialised to linearly decrease to zero at the edge of the cavity. At the stator and the edge of the cavity, a no-slip boundary condition is imposed. The *rotatingWallVelocity* is imposed to the rotor and the shaft to describe the no-slip boundary condition for a rotating patch. For the inlet and the outlet, a linear velocity profile for the tangential component was forced as a boundary condition, such that the tangential velocity is zero at a stationary wall and equal to $\omega_{rot} r$ at a rotating wall. To initialise turbulence, random Gaussian fluctuations were superimposed to the initial velocity profile. This initial field can be consulted in Listing N.13.

Simulation procedure

The flow solver *pimpleFoam* was executed in parallel using *mpirun*. Using *mpirun*, multiple processes can be processed simultaneously on multiple cores, yielding significant decrease in computational time. To run the simulation on multiple cores, the domain should be decomposed such that each domain can be processed on one core. The decomposition method was selected to be the Scotch method, which minimises the number of processor boundaries and interprocessor communication, and consequently the computational time [77]. The simulations were processed on 256 cores on the supercomputer Snellius. In Appendix E, the scalability analysis on the number of cores is given, from which it was determined that 256 cores would lead to the shortest computational time. After the simulation finished, the computational domains were reconstructed to yield to original domain back again.

The time step that was used was selected on basis of the Courant-Friedrichs-Lewy number. The CFL number, as defined in Equation 3.2, should be below 1 for numerical stability [39, 71]. The time step was adapted such that the maximum CFL number on the computational domain remained at a value of 0.2. On basis of the averaged velocity profiles on multiple locations in the rotor-stator cavity system, it was verified whether a steady state was reached. A steady state was confirmed when the temporal gradient of the relative difference with the previous time step starts to oscillate around 0.

$$Co = \frac{u_i \Delta t}{\Delta x_i} \tag{3.2}$$

After the simulation was performed, the energy dissipation rate (ε) was computed from the strain rate tensor ($\boldsymbol{S}_{ij}$). This computation is given in Equations 3.3 and 3.4 [32]. In this equation, ν_{eff} denotes the effective kinematic viscosity, which is the sum of the turbulent eddy viscosity and the molecular viscosity. This equation can be expanded in all Cartesian coordinates and rewritten to Equation 3.5 [78]. Using the definition of the covariance, the relation between the average of the products and the product of the averages was used. An example of this is provided in Equation 3.6. Finally, to get the total energy dissipation rate, the energy dissipation rate as predicted by the sub-grid-scale turbulence model is added to the resolved energy dissipation rate (Equation 3.7.

$$\varepsilon = 2\nu_{eff}\overline{\boldsymbol{S}_{ij}\boldsymbol{S}_{ij}} \tag{3.3}$$

$$\boldsymbol{S}_{ij} = \frac{1}{2}\left(\frac{\partial u'_i}{\partial x_j} + \frac{\partial u'_j}{\partial x_i}\right) \tag{3.4}$$

$$\varepsilon = \nu_{eff}\left(\begin{array}{c} 2\overline{\frac{\partial u'_x}{\partial x}^2} + \overline{\frac{\partial u'_y}{\partial x}^2} + \overline{\frac{\partial u'_z}{\partial x}^2} + \overline{\frac{\partial u'_x}{\partial y}^2} + \\ 2\overline{\frac{\partial u'_y}{\partial y}^2} + \overline{\frac{\partial u'_z}{\partial y}^2} + \overline{\frac{\partial u'_x}{\partial z}^2} + \overline{\frac{\partial u'_y}{\partial z}^2} + \\ 2\overline{\frac{\partial u'_z}{\partial z}^2} + 2\left(\overline{\frac{\partial u'_x}{\partial y}\frac{\partial u'_y}{\partial x}} + \overline{\frac{\partial u'_x}{\partial z}\frac{\partial u'_z}{\partial x}} + \overline{\frac{\partial u'_y}{\partial z}\frac{\partial u'_z}{\partial y}}\right) \end{array}\right) \tag{3.5}$$

$$\mathrm{cov}(\overline{\frac{\partial u'_x}{\partial z}}, \overline{\frac{\partial u'_z}{\partial x}}) = \overline{\frac{\partial u'_x}{\partial z}\frac{\partial u'_z}{\partial x}} - \overline{\frac{\partial u'_x}{\partial z}} \cdot \overline{\frac{\partial u'_z}{\partial x}} \tag{3.6}$$

$$\varepsilon_{tot} = \varepsilon + \varepsilon_{SGS} \tag{3.7}$$

3.2 Simulation of a rs-SDR

The model of the hydrodynamics in a rotor-stator cavity was extended towards a complete rs-SDR with optional throughflow. In order to do this, a computer-aided design (CAD) was constructed in SolidWorks and converted to an OpenFOAM mesh. In the remainder of this section, the methodology of setting up these simulations and performing the analysis is provided.

Numerical implementation

The numerical implementation of the complete rs-SDR simulations is identical to the rotor-stator cavity simulations, apart from the interpolation schemes. These schemes are given in Table 3.3. Compared to the rotor-stator cavity simulations, a higher-order scheme for the divergence terms is used. This yields a higher order of convergence and hence smaller computational time, but reduces the stability compared to the TVD scheme as used in the rotor-stator cavity case. As the computational time of this simulation was found to be rather large, improving the order of convergence was concluded to be of more importance.

Mathematical term	**Interpolation scheme**
Gradient	Gauss cubic
Divergence	Gauss cubic
Laplacian	Gauss cubic corrected
Interpolation	Cubic
Surface-normal gradient	Cubic corrected

Table 3.3: Interpolation schemes that were used in the rotor-stator spinning disc reactor simulations.

Computational mesh

Firstly, SolidWorks was used to make a CAD that resembles the shaft and rotor of a rs-SDR. Using *blockMesh*, a mesh was made shaped like the cylindrical enclosing of the rs-SDR. The mesh on which the flow behaviour is simulated was obtained by subtracting the CAD design of the rotor and shaft from the cylindrical enclosing using the utility *snappyHexMesh*. This geometry can be found in Appendix F, together with the CAD of the rotor and shaft. Refinement was included such that grid cell sizes near the walls were increasingly smaller. The details on the construction of the mesh can be found in the source code in listing N.12. Using the grid dependency study as presented in Appendix G, an adequate resolution was determined. To conclude, this resulted in a mesh consisting of $1.5 \cdot 10^6$ cells with a maximum non-orthogonality of $0.14°$.

Initialisation and boundary conditions

The velocity field was initialised on basis of the inlet flow rate and the rotational rate of the disc. Using a velocity of 40% of the local rotor velocity, as discussed previously, the tangential velocity was computed at each radial position. On the basis of continuity, the radial velocity was computed at each radial position. Thereafter, the radial and tangential velocity were decomposed into Cartesian coordinates for implementation into OpenFOAM. The pressure field was initialised to be *uniform 0* and the turbulent viscosity was uniformly initialised using the kinematic viscosity. On the walls, a no-slip boundary condition was applied for the velocity and a zero-gradient boundary condition for the pressure. The *rotatingWallVelocity* was imposed to the rotor and the shaft to describe the no-slip boundary condition for a rotating patch. For the turbulent viscosity, *nutUSpaldingWallFunction* was applied at all walls, to lessen the importance of the constraints on y^+ and r^+, especially. At the inlet, a *fixedValue* boundary condition was forced for the velocity, which ensures a constant inlet flow rate. The pressure boundary condition was zero-gradient at this location. At the outlet, the pressure was fixed at the reference pressure and the velocity was modelled using an *inletOutlet* boundary condition. This boundary condition computes the velocity using a zero-gradient boundary condition, but sets the velocity to zero in case of an inward velocity. For the turbulent viscosity, the *calculated* boundary condition was used.

Simulation procedure

The flow solver *pimpleFoam* was executed in parallel using *mpirun* on 256 cores on the Dutch supercomputer Snellius. The mesh was decomposed using the Scotch method. Adaptive time-stepping was implemented to ensure a CFL number of 0.2. On basis of the averaged velocity profiles, it was verified whether a steady-state was reached. A steady state was confirmed when the temporal gradient of the relative difference with the previous time step starts to oscillate around 0. For all simulations, a steady state was reached after approximately half a residence time.

Residence time distribution modelling

After a steady-state velocity field was obtained from the simulations, the residence time distribution was estimated from this field. The solver *scalarTransportFoam* was rewritten to describe the convection and diffusion of a passive scalar (f) in a velocity field. The equation that is solved is given in Equation 3.8. In this simulation, the velocity field and the turbulent diffusion coefficient were obtained from the previously simulated velocity field and local turbulent viscosity, using a turbulent Schmidt number of 0.9. For the molecular diffusion coefficient of the passive scalar, a value of $1 \cdot 10^{-9}$ m^2 s^{-1} was used. At the outlet, the mixing cup concentration is computed in each time step. Since the inlet concentration of the passive scalar is set to unity as a step input, the time derivative of the mixing cup outlet concentration represents the residence time distribution function. For more simulation details, the source code in Listings N.34 and N.35 can be consulted.

$$\frac{\partial f}{\partial t} + \nabla \cdot f\boldsymbol{u} = \nabla \cdot (D_m + D_t) \nabla f \tag{3.8}$$

3.3 Simulation of bacterial-resolved flow

3.3.1 Simulation of a turbulent Couette flow

In order to study the effect of shear and turbulence on the bacterium *Clostridium autoethanogenum*, a turbulent Couette flow field is simulated. This flow field resembles the flow in the rs-SDR in terms of shear and turbulent kinetic energy, but requires significantly less computational costs to simulate due to the absense of rotational effects. In this section, the methodology of simulating turbulent Couette flow is outlined. Two cases are simulated. For one case, which acts as a validation case, the bottom wall and top wall move in opposite directions with equal velocity. More details on this validation case can be found in Appendix A. In the other case, which will be used for the bacterial deformation simulations, the bottom wall is stationary and the top wall moves.

Numerical implementation

The interpolation schemes that were used in this simulation are given in Table 3.4. In all cases, linear interpolation schemes were selected, as these were found to be non-dissipative, which is important to capture the turbulent character of the flow [79]. Time discretisation was selected to be Crank-Nicolson with a blending coefficient of 0.9. The solvers are selected analogously to the previous simulation cases. For pressure-velocity coupling, PISO was selected with 2 correctors. The LES sub-grid-scale model was selected to be WALE with a cutoff width *cubeRootVol*.

Mathematical term	**Interpolation scheme**
Gradient	Gauss linear
Divergence	Gauss linear
Laplacian	Gauss linear corrected
Interpolation	Linear
Surface-normal gradient	Orthogonal

Table 3.4: Discretisation schemes that were used in the simulations of a turbulent Couette flow.

Computational mesh

The mesh was constructed using *blockMesh* from one block with hexahedral cells. The mesh is a rectangular cuboid. In the wall-normal direction, non-uniform grid spacing is used, with an expansion coefficient of 200. This was done to ensure wall resolving and place grid cells in the viscous sublayer. The mesh sizes that were used in the simulations for the validation case and the Couette flow field simulation that is to be used in the bacterial cell deformation simulation are listed in Table 3.5. The number of grid cells was selected to be N_x x N_y x N_z = 50 x 110 x 50 after a grid dependency study was performed on the validation case, which is given in Appendix A.

	Validation case	**Bacterial deformation flow field**
Length [m]	0.05	$1 \cdot 10^{-3}$
Width [m]	0.05	$2 \cdot 10^{-4}$
Height [m]	0.05	$1 \cdot 10^{-3}$

Table 3.5: Mesh sizes that were used in the simulations of a turbulent Couette flow.

Initialisation and boundary conditions

Only the boundary conditions on the walls are discussed, since the boundaries on the sides are treated as cyclic boundaries to resemble an infinite plate case. For the pressure field, the wall boundary conditions was selected to be *zeroGradient* and the initial condition was set to 0. For the turbulent viscosity, Spalding's wall function was used to force the correct turbulent viscosity profile near the wall. The kinematic viscosity was used as the initial condition. The boundary conditions for the velocity at the walls was set to a no-slip condition forcing the wall velocity. The initial velocity field was acquired by running a simulation with initially a linear velocity profile with random Gaussian perturbations in all directions at a high Reynolds number to ensure a turbulent initial velocity field.

Simulation procedure

The flow solver *pisoFoam* was executed in parallel using *mpirun* on 16 cores. The decomposition was performed using the Scotch method. The time step was selected such that the maximum CFL number remained at 0.2. On basis of the averaged velocity profiles, it was verified whether a steady state was reached. A steady state was confirmed when the temporal gradient of the relative difference with the previous time step starts to oscillate around 0.

3.3.2 Bacterial cell simulation

To study the effect of shear and turbulence effects on the bacterial cell, a bacterial cell is simulated in a turbulent Couette flow of which the simulation details were previously discussed. In this turbulent Couette flow, the bottom wall is stationary and the top wall moves in a uniaxial direction, to resemble the flow in a rotor-stator spinning disc reactor. Analogous to the deformation investigations on red blood cells [34, 35, 37], the VoF method was implemented to model the bacterial cells. The literature review on bacterial properties in Section 2.4 was used to determine simulation parameters. In the remainder of this section, the details on the numerical implementation and simulation are discussed.

Numerical implementation

The numerical schemes that were used are given in Table 3.6. In most cases, standard linear interpolation schemes were used. For the discretisation of the divergence terms, different schemes were implemented for each term. In order to increase the numerical stability, the limited scheme *LimitedLinearV 1* was implemented. This scheme limits towards upwind discretisation in regions with a rapidly changing gradient. The limiter is calculated on basis of the component of the velocity vector in which the gradient changes the most rapidly. This increases stability, but reduces accuracy [70]. In order to accurately capture the interface in the VoF method, the velocity-weighted multicut piecewise linear interface calculation (MPLICU) scheme was implemented. This scheme is especially more accurate in cases of high shear, as it uses velocities at the faces in order to compute the face flux. Additionally, this scheme is more precise for meshes with refinement. Both these aspects make this scheme suitable for the bacterial simulations. Time discretisation was selected to be a complete Crank-Nicolson scheme. In contrast to previous simulations, the Smagorinsky-Lily sub-grid-scale turbulence model was selected, as WALE is incompatible with multiphase solvers in OpenFOAM.

The solvers used in the simulation were selected analogously to tutorial cases. For pressure, *PCG* was selected with a *DIC* smoother. The hydrostatic pressure was solved using *GAMG* with DIC as a smoother. All other solvers were selected to be *smoothSolver* with a *symmGaussSeidel* smoother. The volume fraction was solved with an interface compression coefficient (C_α) of 1. The pressure-velocity coupling was solved using PIMPLE with 2 corrector loops and 4 non-orthogonal corrector steps. Other numerical settings can be found in Listing N.24 and N.25.

Mathematical term	**Interpolation scheme**
Gradient	Gauss linear
Divergence (default)	Gauss linear
$\mathrm{div}(\rho \cdot \phi, \boldsymbol{U})$	Gauss limitedLinearV 1
$\mathrm{div}(\phi, \alpha)$	MPLICU
Laplacian	Gauss linear corrected
Interpolation	Linear
Surface-normal gradient	Corrected

Table 3.6: Discretisation schemes that were used in the simulations of a bacterial cell.

Computational mesh

The same base mesh was used as discussed for the turbulent Couette case. To decrease the maximum CFL number in these simulations, a moving mesh was introduced that moves with half of the wall velocity. In order to model the flow on a bacterial scale, the base mesh was refined. This refinement was performed gradually throughout the domain. In 10 steps, a box with decreasing size around the location where the bacterial cell is to be initialised is refined by bisecting the cells in each direction. Using this procedure, a sufficiently refined mesh was obtained to resolve the bacterium. This yields a total number of grid cells of $1.3 \cdot 10^6$. The quality of the mesh was checked using *checkMesh*, which showed an average non-orthogonality of 6° and a maximum aspect ratio of 38. The maximum non-orthogonality was found to be 68°. From this, it was concluded that the mesh is adequate for simulation, but non-orthogonal correctors are required.

Initialisation and boundary conditions

Similar to the turbulent Couette case, cyclic boundaries on the sides were applied. On the walls, a *zeroGradient* boundary condition was used for the (hydrostatic) pressure. For the turbulent viscosity, a *calculated* 0 boundary condition was used. The kinematic viscosity was used as the initial condition. For the velocity, a *fixedValue* boundary condition is applied to mimic the no-slip condition. The velocity of the top wall was calculated using the specified Reynolds number on basis of the channel width and water viscosity. As an initial field, the turbulent Couette flow field as described in Section 3.3.1 was used. Using the utility *funkySetFields*, the volume fraction of the bacterial cell was set to unity using a mathematical expression for the shape of the bacterial cell. The cell shape was described by a cylinder with two ellipsoid caps. More details on the geometry of the bacterium can be found in Appendix C. The implementation of this geometry in the OpenFOAM simulation can be found in Listing N.27. Near the bacterial cell, the velocity fluctuations were filtered out close to and inside of the bacterial cell in order to avoid instantaneous deformation during the initialisation of the model.

Physical models

Multiple physical models were implemented in the multiphase solver *multiphaseInterFoam*. Firstly, the surface tension force was modelled by the CSF model with a surface tension between the bacterial and water phase equal to 0.0056 N/m on basis of the previously discussed literature. The correlation for drag of both phases was selected to be the Schiller-Naumann correlation. The isothermal diameter model was used for the bacterial phase. Since the Schiller-Naumann correlation is strictly only valid for spheres, the results should be studied with this in mind. Virtual mass is taken into account by means of the *constantCoefficient* model. Finally, gravity is also included in the model.

The deformation of the bacterium was modelled using a viscoelastic model. The Herschel-Bulkley model was implemented to capture the highly viscous character of the bacterium at low shear rates and yielding at higher shear rates. The Herschel-Bulkey model describes fluids that behave as rigid solids with high viscosity at local shear stresses lower than the yield stress, but start to flow as non-linear viscous fluids when this yield stress is exceeded [80]. The implementation of the Herschel-Bulkey model in OpenFOAM is given in Equation 3.9. In this equation, ν_0 is the kinematic viscosity at low shear rates, τ_0 is the density-normalised yield stress and k_{HB} is the consistency coefficient. The order, n_{HB}, was set to unity to represent Newtonian behaviour. The values for the yield stress, kinematic viscosity and consistency coefficient were selected on basis of the previously discussed literature on the mechanical properties of bacterial cell walls.

$$\nu = \min(\nu_0, \frac{\tau_0}{\dot{\gamma}} + k_{HB}\dot{\gamma}^{n_{HB}-1}) \tag{3.9}$$

Simulation procedure

The flow solver *multiphaseInterFoam* was executed in parallel using *mpirun* on 4 cores. The decomposition was performed using the Scotch method. The time step was selected such that the maximum CFL number remained between 0.2 and 0.3. In these simulations with an interface compression velocity, the CFL should also be evaluated on basis of the interface compression velocity. It was found that the CFL number related to the interface compression was not limiting for all cases. For all cases, a simulation time of approximately 0.2 eddy turnover times was ensured.

3.4 Simulation of particle-laden rs-SDR flow

In this section, the simulation of bacteria-laden flow inside the rs-SDR is discussed. For this, an Euler-Lagrange simulation was set up. Firstly, the type of coupling was determined according to the regimes of Elghobashi [46], as depicted in Figure 2.3. Using a biomass concentration of 0.3 g/L, the volume fraction was determined to be such that two-way coupling is necessary. Euler-Lagrange simulations that track each particle individually are more fundamental applications, but the associated computational costs limits its application to small domains or low numbers of particles [81]. The multiphase particle-in-cell (MP-PIC) method was developed to simulate particle-laden flows that can range from dilute to dense loadings and variable particle sizes [82]. The method is based on capturing multiple particles into a computational particle. The dynamics of the real particle phase can then by extracted by solving a Liouville equation for the particle distribution function. This methodology is discussed in Appendix L. In this way, accurate mapping from Lagrangian particles to the Eulerian grid is enabled [82]. Using this model, multiphase flows can be simulated that would otherwise require excessive computational time with reasonable accuracy [82, 83]. The model has been applied to particle-laden flows such as flows inside a hydrocylone [81], circulating fluidized beds [84] and rotating drums [85]. In this section, the implementation of this model to bacteria-laden flow in the rs-SDR is discussed. It must be noted that this section is by no means an extensive review of the MP-PIC method and the reader is referred to Andrews et al. [82] for more theoretical details.

Numerical implementation

For the numerical implementation of this model, the solvers and discretisation schemes were implemented as discussed in Section 3.2 for the simulation of the complete rs-SDR. As an addition, *limitedLinear* numerical schemes were added to ensure boundedness of the volume fractions. These numerical schemes can be consulted the source code in Listing N.32. In contrast to previous simulations, the Smagorinsky-Lily sub-grid-scale turbulence model was selected, as WALE is incompatible with multiphase solvers in OpenFOAM. The computational mesh and the geometry of the rs-SDR as used in this simulation is the same as the mesh as presented in Section 3.2. As initial conditions of the flow inside the rs-SDR, the mean results of the LES simulations were used, which were averaged over 25 rotations. The same boundary conditions were maintained.

Physical models

As discussed previously, the fluid-particle interactions were solved in a coupled manner using the MP-PIC model. In *cloudProperties*, all details on the particle phase can be specified. Firstly, the particle forces were specified. In this model, the gravity, drag, virtual mass, pressure gradient and lift force were included. For the virtual mass force, a virtual mass force constant of 0.5 was included. For the drag force, the non-spherical drag correlation by Haider and Levenspiel was implemented, since this predicts the drag on objects with a sphericity similar to the bacterium with reasonable accuracy [86]. For the lift force, the Saffman-Mei correlation is used, which considers shear-induced lift [87]. It must be noted that this correlation was strictly only applicable to spherical geometries, whereas the bacteria are rod-like. Therefore, the results of this simulation should be treated with this caveat in mind. More details on the particle forces can be consulted in Appendix L

The computational particles were injected using the *patchInjection* model and the number of these parcels was determined to be 1 million per second. The injection velocity of the particles was set equal to the inlet velocity of the fluid. The particle size distribution was set to a fixed value equal to the bacteria length. The interaction with the walls was described by a rebound with an arbitrarily selected elasticity coefficient 0.97 of and a restitution coefficient of 0.09. However, the influence of these parameters seems limited due to the low number of interactions with the walls. For the inter-particle interactions, the implicit packing model was selected using the Harris-Crighton particle stress model. A linear dependency of the stress and the particle fraction was assumed. The close-packed volume fraction of the bacteria was selected to be 0.4 on basis of a study on packing of rod-shape cylinders with varying aspect ratios [88].

Simulation procedure

The solver *denseParticleFoam* was executed in parallel using *mpirun* on 256 cores on Snellius. The decomposition was performed using the Scotch method. The time step was selected such that the maximum CFL number remained between 0.2 and 0.3. The particles were injected continuously and uniformly in time throughout the simulation time, which was selected to be one residence time.

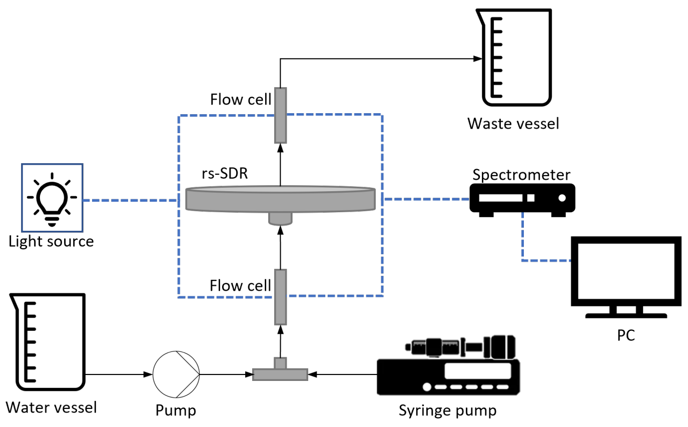


Figure 3.2: A schematic representation of the setup used for the macromixing characterisation of the rs-SDR.

3.5 Residence time distribution experiments

In order to validate the rs-SDR simulations, the macromixing behaviour in the rs-SDR was studied. In Figure 3.2, a schematic overview is given for the setup that was used to determine the residence time distribution in the rs-SDR. For this, water was passed through the bottom of the rs-SDR to the top of the reactor, after which it was disposed in a waste vessel. The flow was maintained using a VerderGear VG1000 gear pump. Using a KD Scientific syringe pump, a methylene blue tracer solution ($4.69 \cdot 10^{-2}$ M) was injected into the flow at a flow rate of $1.39 \cdot 10^{-7}$ m^3 s^{-1}. At the inlet and outlet, 10 mm Avantes flow cells were used to monitor the absorption spectra using optical fibres. The light source used was an AvaLight-DHS, which produces light with a wavelength over a range of 191 nm to 750 nm. The spectrometer (AvaSpec-DUAL) measured the number of counts with an integration time of 1.1 ms. A PMMA rs-SDR was used, which has the same dimensions as implemented in the model, as specified in Appendix F. The inlet flow rate was set to $1 \cdot 10^{-5}$ m^3 s^{-1} and the rotational speed was varied from 71 to 1145 RPM. The absorption of the tracer was measured at 613 nm. Using Beer-Lambert's law and a calibration curve, the absorbance was converted to tracer concentration. These calibration curves can be consulted in Appendix H. After injection, the outlet signal was measured for at least 10 residence times. Each experiment was performed three times.

To take into account relatively long injection time with respect to the residence time, a time-domain deconvolution was used [89]. For this, the outlet signal was described as a convolution between the inlet signal and the reactor model, as described in Equation 3.10. Using MATLAB's *lsqnonlin*, the model parameters of the engineering model (Equation 2.39) were fitted on basis of the error between the measured outlet concentration and the predicted outlet concentration. Using this method, high R^2 values (≥ 0.99) can be obtained [89].

$$c_{out}(t) = \int_0^t E(t) c_{in}(t-\tau) dt \tag{3.10}$$

Chapter 4
Results and Discussion

4.1 Rotor-stator cavity simulations

To validate the simulation of the rotor-stator cavity system, a simulation case was set up that corresponds to the system that was experimentally studied and modelled using a Reynolds stress model (RSM) by Poncet et al. [57]. In Figure 4.1, the axial profiles of the averaged tangential and radial velocity are plotted for different radial positions. The value of z^* was defined to be 0 at the rotor and 1 at the stator. The velocities are normalised by the local linear disc velocity. It can be seen that the correspondence with the experimental results is adequate for the tangential velocities. For the radial velocity profile, the correspondence with the experimental results adequate at low radial positions, but at higher radial positions, the model deviates significantly from the experimental data. In Figure 4.2, the normal components of the Reynolds stresses are compared with experimental and simulation results from Poncet et al. [57]. The Reynolds stresses were normalised with the square of the local disc velocity. The Reynolds stresses that resulted from the simulations in this study consist of both the resolved and unresolved part of the Reynolds stresses, which were both time-averaged and summed to obtain the total Reynolds stress tensor. From Figure 4.2, it can be seen that there are strong deviations between the experimental data and the simulation results. At low radial positions, the correspondence with the experimental data is better than the RSM model, but the correspondence is worse at higher radial positions.

The entrainment coefficient, which is defined as the ratio between the tangential velocity in the core of the fluid and the tangential velocity of the disc at that radial position, was studied by Poncet et al. [57]. In this study, a correlation was found for the entrainment coefficient as a function of the local flow rate coefficient. A limiting entrainment coefficient of 0.43 is predicted at zero superimposed throughflow. From the velocity profiles as depicted in Figure 4.1, it was found that the entrainment coefficient is 0.39 at midradial position, which deviates slightly compared to the experimental correlation. However, the entrainment coefficient does closely match with the experimental and simulation results by Poncet et al. [57] at lower radial position. At higher radial positions, the underestimation of the entrainment coefficient is more prominent when compared to the results by Poncet et al. [57].

In Figure 4.3, the axial profile of the energy dissipation rate and the turbulent kinetic energy in the rotor-stator cavity are visualised. From this figure, it can be seen that most of the turbulent kinetic energy is present in the turbulent boundary layers, with the largest turbulent kinetic energy in the Bödewadt layer. Next to this, it was found that the energy dissipation rate in this boundary layer is much larger than in the inviscid core of the flow and near the rotor. A similar distribution of the turbulent kinetic energy was reported by Poncet et al. [56], who reported a maximum normalised turbulent kinetic

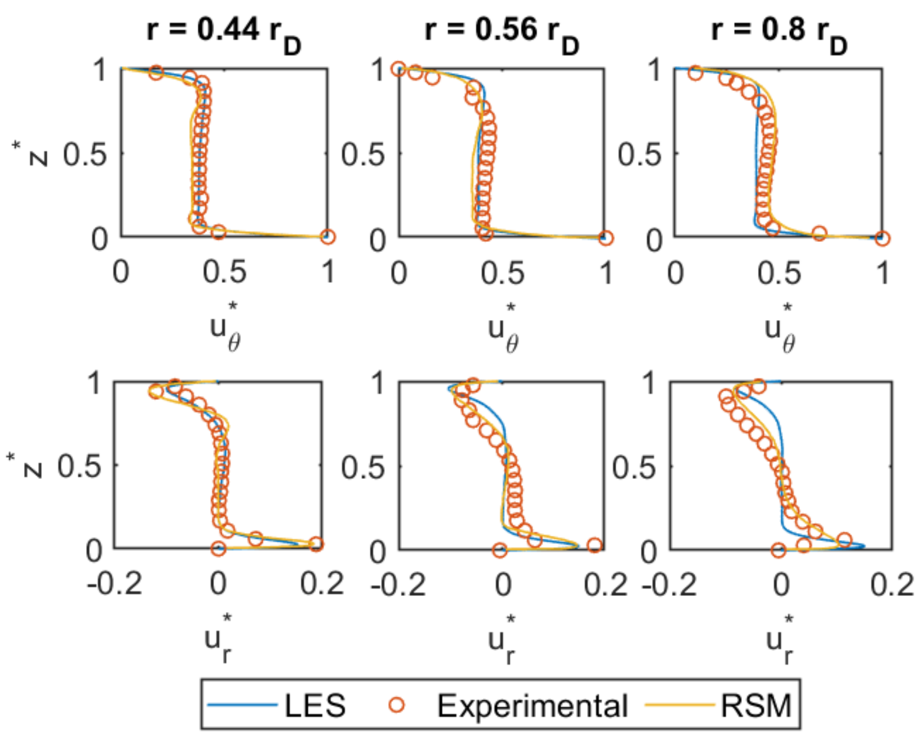


Figure 4.1: Comparison of the axial velocity profiles with the validation case by Poncet et al. [57]. The red circles denote the experimental data by Poncet et al., the yellow line denotes the RSM simulation data by Poncet et al. and the blue line denotes the simulation results from this study.

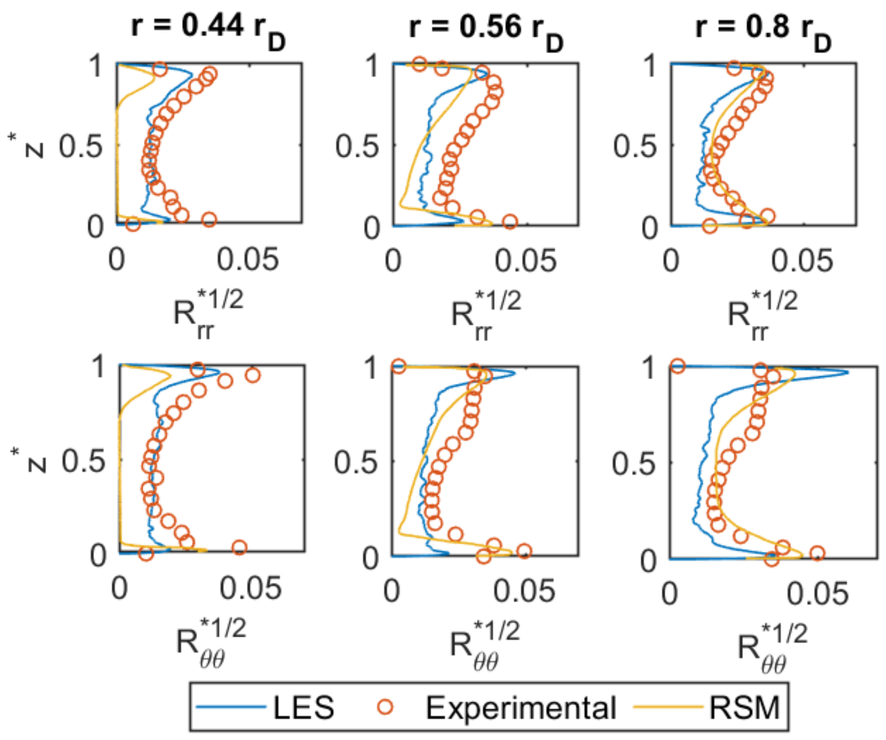
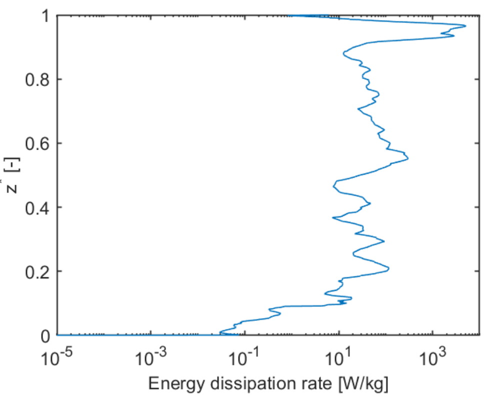
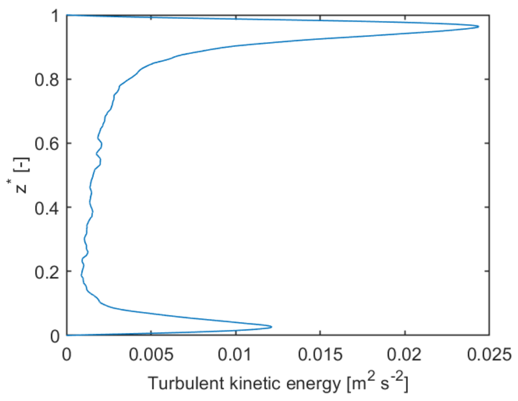


Figure 4.2: Comparison of the Reynolds stress profiles with the validation case by Poncet et al. [57]. The red circles denote the experimental data by Poncet et al., the yellow line denotes the RSM simulation data by Poncet et al. and the blue line denotes the simulation results from this study.



(a) Energy dissipation rate



(b) Turbulent kinetic energy

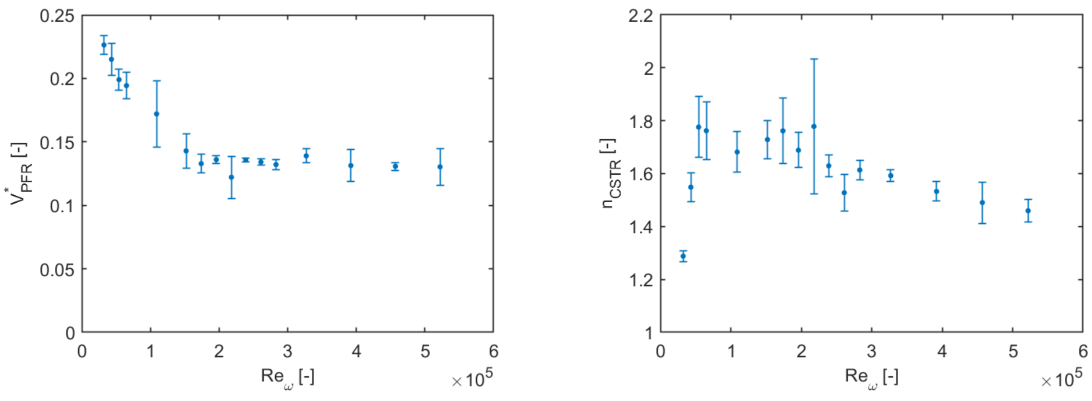
Figure 4.3: Axial profiles of turbulent characteristics of the flow in the rotor-stator cavity at a radial position of $0.8r_d$.

energy ($k^* = k/(\omega r_o)^2$) of $9 \cdot 10^{-3}$, which resembles the averaged value of $1 \cdot 10^{-2}$ at a radial position of 99.5% of the disc radius in this study.

4.2 rs-SDR simulations

The simulation of the rotor-stator cavity was extended towards simulating the complete rotor-stator spinning disc reactor. The flow in the rs-SDR was simulated for four different rotational Reynolds numbers. In this section, the results of these simulations are discussed.

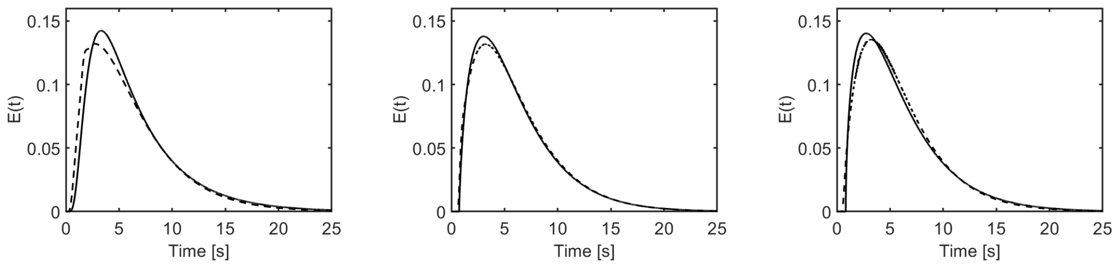
To validate the flow fields in the rs-SDR, the residence time distribution of the rs-SDR was studied experimentally and numerically from the CFD simulations. Using the engineering model for a rs-SDR (Equation 2.39), these RTDs can be characterised by a number of tanks-in-series and a volume fraction that corresponds to the PFR-zone. In Figure 4.4, these characteristics are visualised for the experimentally obtained RTDs. It can be seen that with increasing rotational Reynolds number, the volume fraction of the PFR-zone and the number of tanks-in-series decreases, which is related to the increase of the rotation governed zone in the rs-SDR. At a rotational Reynolds number below $1 \cdot 10^5$, a different trend was observed, as the transition towards the laminar regime occurs at this Reynolds number. In this regime, the volume fraction of the PFR-zone rapidly increases and the number of the tanks-in-series decreases. These trends are in correspondence with results from literature [18, 89].



(a) Volume fraction of PFR-zone

(b) Number of tanks-in-series

Figure 4.4: The characteristics of the RTD in the rs-SDR according to the engineering model for the experimentally obtained RTDs.



(a) 25 rad/s

(b) 50 rad/s

(c) 75 rad/s

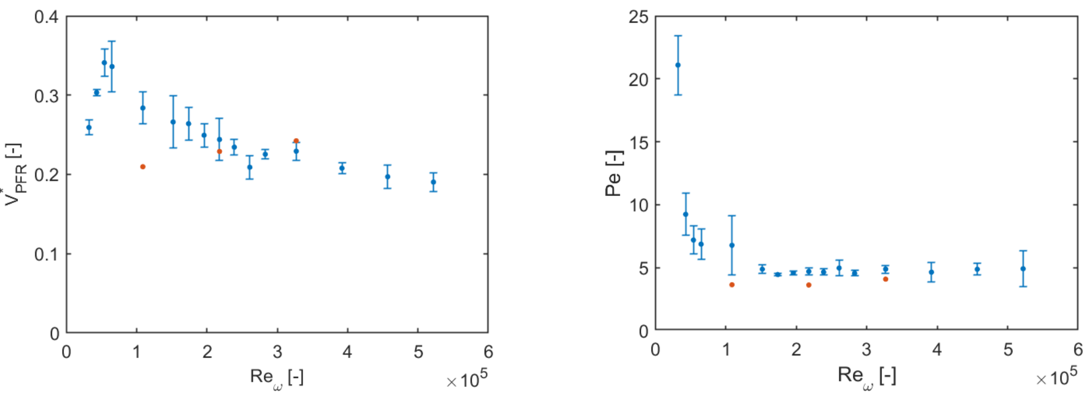
Figure 4.5: The experimentally obtained residence time distributions using deconvolution with the engineering model compared with the simulation results. The solid lines denote the experimental results and the dashed line denote the simulation results.

In Figure 4.5, the experimental residence time distributions that were deconvoluted using the engineering model are compared with the simulated residence time distributions. For the simulations that were performed in the turbulent regime, the correspondence with the simulation results and the experimental RTD is generally well. The simulation in the laminar regime did not correspond well with the experimental RTD, as the simulated RTD was found to have less dispersion as in the experimental result, yielding a RTD that resembles a PFR and CSTR in series. The relatively small deviation between the simulated and experimental RTD in the turbulent regime can be related to the dispersion in the PFR-zone that is present in the simulations, but which is not considered in the engineering model. To take into account this dispersion, a different engineering model is proposed, which describes the macromixing behaviour in the rs-SDR by means of a non-ideal PFR with axial dispersion and a CSTR in series. In Equation 4.1, the response of a pulse injection for a non-ideal PFR is given. This response was numerically used as inlet condition for a CSTR. The derivation of this equation is elaborated upon in Appendix I.

$$c_{out}(t) = \left(Pe\frac{\tau_{PF}}{4\pi t}\right)^{0.5} \exp\left(-Pe\frac{(1-t/\tau_{PF})^2}{4t/\tau_{PF}}\right) \tag{4.1}$$

This proposed reactor model was used in the deconvolution to study the RTD from the experimental results. In Figure 4.6, the fitted volume fraction of the PFR-zone and the

Péclet number are depicted as a function of the rotational Reynolds number for both the experimental data as well as the simulation results. In the turbulent regime, it can be seen that the volume fraction of the throughflow governed zone decreases with increasing rotational Reynolds number, but the Péclet number remains constant. In the laminar regime, the Péclet number increases due to the lower dispersion and the volume fraction of the PFR-zone decreases. The simulations results in the turbulent regime correspond well to the experimental results, however the Péclet number is underestimated, which can be related to additional numerical dispersion in the simulations. In Appendix I, more details are provided on the fitting of the engineering model and the proposed non-ideal model, together with the simulation of the residence time distribution.



(a) Volume fraction of PFR-zone (b) Péclet number

Figure 4.6: The characteristics of the RTD in the rs-SDR using deconvolution with the proposed non-ideal model on the experimental data and the simulation results.

In Figure 4.7, the axial profiles of the mean radial velocity are visualised for different radial positions. The value of z^* was defined to be 0 at the rotor and 1 at the stator. It can be seen that at low radial positions, a throughflow governed velocity profile is present, whereas the rotation governed velocity profiles are present at higher radial positions. The throughflow governed zone can be allocated to the PFR zone in the rs-SDR and the rotation governed zone behaves as a (or multiple) CSTR(s) due to the intense recirculation. To connect these velocity profiles to the overall macromixing behaviour in the rs-SDR, the radial position at which this transition occurs was compared with Equation 2.38. The transition radius for the simulation results was defined as the radius at which the axial derivative of the radial velocity at the rotor is zero. At this radial position, the onset of the recirculation in the Bödewadt boundary layer starts. In Figure 4.8, the simulated and experimental results of the transition radius are compared to Equation 2.38. The experimentally determined transition radii are found to follow the correlation by De Beer [19] generally well, meaning that the transition radius increases with increasing dimenionless parameter $(C_W Re_\omega^{-4/5})^{5/13}$. It can be seen that the transition radius is underestimated by the simulation results, which is related to the definition of the transition radius from the velocity profiles. To be considered as a well-mixed zone in the engineering model, a higher degree of recirculation is required. Therefore, the transition radius on basis of the velocity profiles is lower than the transition radii that were experimentally observed, yet it can be seen that the scaling law as function of the dimensionless parameter $(C_W Re_\omega^{-4/5})^{5/13}$ is similar. In Appendix J, more details are provided on the recirculation in the rs-SDR.

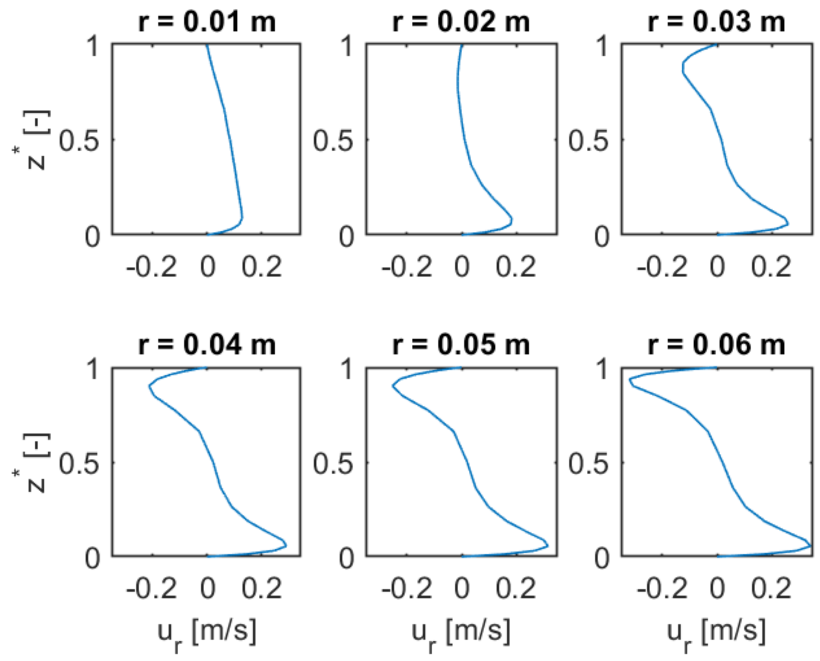


Figure 4.7: Radial velocity profiles in the rs-SDR model (Re = $2.18 \cdot 10^5$ and $\phi_v = 1 \cdot 10^{-5}$) at different radial positions that exemplify the transition from a throughflow governed regime to a rotation governed regime.

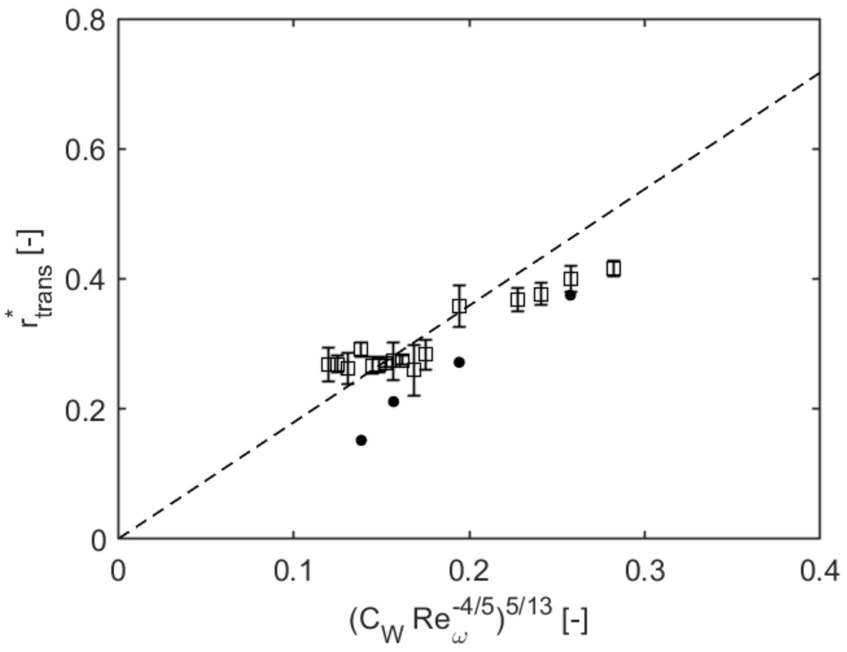


Figure 4.8: Transition radius predicted from the velocity profiles compared to the prediction by De Beer [19]. The squares represent the experimental results as predicted using the engineering model on the RTDs.

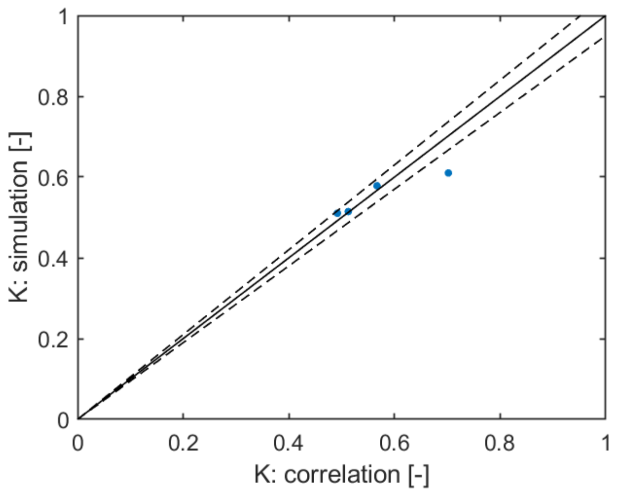


Figure 4.9: Parity plot for the simulated entrainment coefficient and the entrainment coefficient determined for by the correlation as depicted in Equation 4.2. Five percent deviation lines are included.

The entrainment coefficient was determined as a function of the radial position for the simulated Reynolds numbers. For the top gap at which centripetal throughflow is present, the entrainment coefficient was compared with the correlation for the entrainment coefficient as function of the local throughflow coefficient (Cq_r). This correlation, together with the definition of the local throughflow coefficient are depicted in Equation 4.2 [56].

$$K = 2\,(5.9Cq_r + 0.63)^{5/7} - 1 \qquad \text{with} \qquad Cq_r = \frac{Q(\Omega r^2/\nu)^{0.2}}{2\pi r^3 \Omega} \tag{4.2}$$

In Figure 4.9, the entrainment coefficient for the different simulated rotational speeds are compared to the prediction with the correlation by Poncet et al. [56]. To reduce inlet and outlet effects, the entrainment coefficient was determined at mid-radial position. For the three simulations with the highest Reynolds number, the simulation results correspond well to the correlation and the deviation remains between the five percent deviation lines. The maximum deviation was found at the lowest simulated Reynolds number, since this simulation is in the laminar regime of the rs-SDR and the correlation is only applicable to turbulent flows of Batchelor type.

When it is assumed that the flow is of a Batchelor type and that the boundary layers are separated, the thickness of both the Von Kármán and Bödewadt boundary layer is expected to scale with the Reynolds number according to Equation 4.3 [90]. The boundary layer thickness was defined for the Bödewadt layer as the distance from the stator at which the axial derivative of the azimuthal velocity is zero. To remain in the Batchelor flow regime, the boundary layer thickness was determined at a radial position of 0.06 m. The thickness of the Von Kármán boundary layer was defined as the axial position at which the azimuthal velocity has developed for 99%. In Figure 4.10, the boundary layer thicknesses from the simulation results are given. For both the Von Kármán and the Bödewadt boundary layer, the thickness follows the scaling law accurately with R^2-values of 0.997 and 0.977 for the Bödewadt and Von Kármán layer, respectively. The thickness of the Bödewadt boundary layer is larger than the Von Kármán boundary layer and, in contrast to the Von Kármán boundary layer, does not approach zero at infinite Reynolds number. This was hypothesised to be related to the superimposed throughflow, which causes an increase

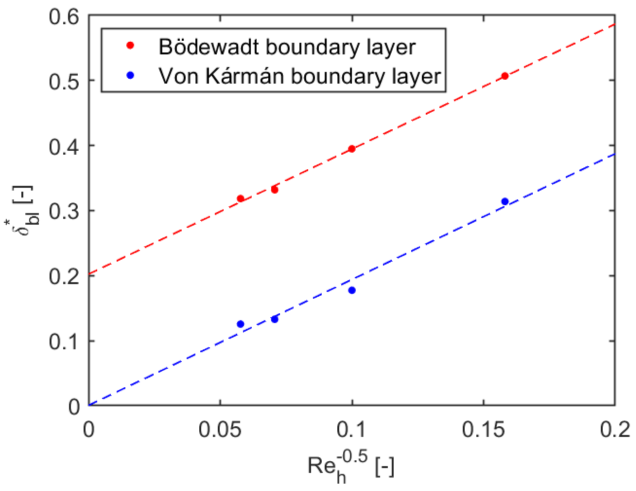


Figure 4.10: The thicknesses of the Von Kármán and Bödewadt boundary layers at half disc-radius in the rs-SDR with a superimposed throughflow. A trendline was included to demonstrate the scaling relationship as shown in Equation 4.3.

in the boundary layer thickness, in contrast to cases without superimposed throughflow where the boundary layer thickness does decrease to zero at infinite Reynolds number [90].

$$\delta_{bl} \propto \frac{1}{\sqrt{Re_h}} \quad \text{with} \quad Re_h = \frac{\Omega h^2}{\nu} \tag{4.3}$$

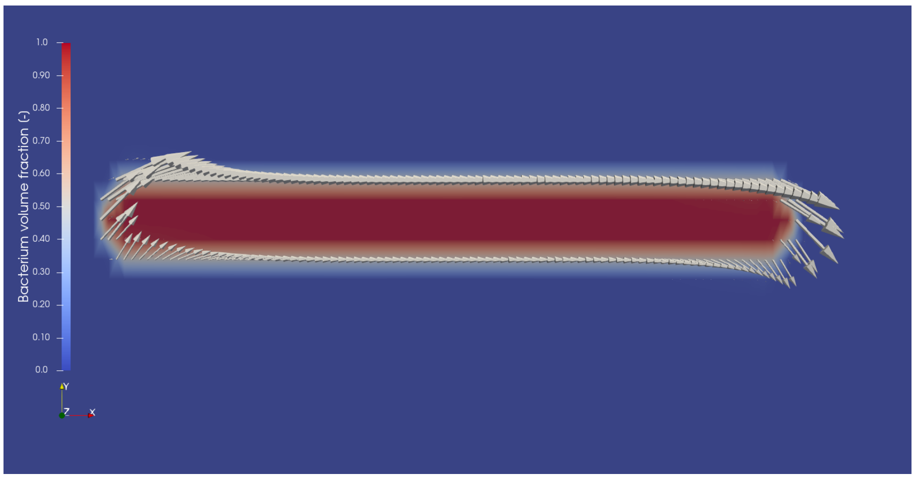
4.3 Bacterial-resolved flow

The results of the simulation of bacterial-resolved flow are discussed in this section. Firstly, the results of the turbulent Couette flow field are discussed. The methodology for simulating a Couette flow field was successfully validated by means of literature data by Pirozolli et al. [91]. Besides, a grid dependency study was performed and a sufficiently grid-independent velocity field was obtained. These results, accompanied by further elaboration, can be reviewed in Appendix A. Subsequently, the Couette flow fields that acts as an initial field for the bacterial-resolved flow simulations were simulated. In Appendix B, the results of the flow fields at the selected Reynolds numbers are presented. As a verification case, a RAS simulation was performed and similar results were obtained.

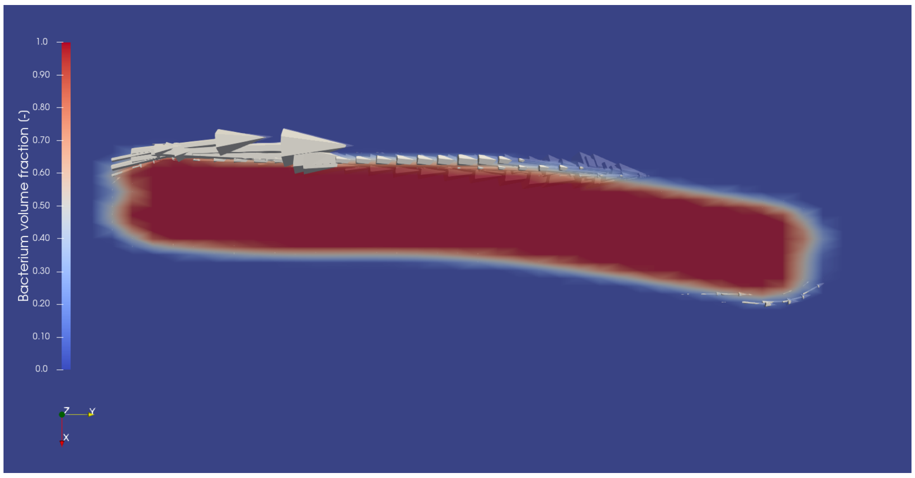
Using these Couette flow fields, the effect of shear and turbulence on the bacterium were studied for three Reynolds numbers. Additionally, the orientation at which the bacterium is initialised was set to spanwise, streamwise and wall-normal to study its effect on the bacterium dynamics. It was found that the drag force acting on the bacterial cell is the dominant force. In Figure 4.11, the orientations of the drag forces acting on the bacterial cell can be found for the lowest simulated Reynolds number. Figures of the orientation of the drag forces on the bacterium for the other Reynolds numbers can be found in Appendix D. It can be seen that these forces will cause a rotation in the shear plane, which is an effect of shear on elliptical cylinders that was previously observed both in experiments and numerical simulations [92–94]. For details on the drag force calculation and the resulting torque, Appendix K can be consulted. In Figure 4.12, the magnitude of the drag force and torque are visualised for the tested Reynolds numbers and bacterium orientations. It can be seen that the drag force magnitude increases with increasing Reynolds number. Next to this, the orientation has an effect on the force magnitude. When oriented streamwise, the bacterial cell experiences little forces due to the small flow-normal surface area. When the bacterial cell is placed in a spanwise orientation, the force magnitude increases, as the flow-normal area increases. Additionally, the bacterium experiences more shear forces due to the spanwise velocity gradients present in the Couette flow field. The maximum force was obtained in a wall-normal orientation, since the shearing effect is larger as the bacterium experiences a larger velocity difference.

To demonstrate the rotation that these forces cause, the torque was computed. Similar trends as discussed for the drag force can be observed. In most of the simulated cases, the orientation of the torque in the spanwise direction was found to be dominant, which results in a rotation in the shear plane, as was visualised in Figure 4.11a. When the bacterium was initialised in a spanwise orientation, the torque in the spanwise direction was found to be less significant compared to torque in the streamwise direction. This can be related to the small lever arm vector for rotation in the shear plane and the presence of velocity gradients in the spanwise direction in the Couette flow field. From the results as presented in Figure 4.12, predictions can be made of the preferential allignment of the bacterial cells. Since the torque on the bacterial cell is the smallest in a streamwise orientation, the angular acceleration will be small. When the torque imposed by the drag force surpasses the viscous torque, the bacterial cell can rotate towards a wall-normal orientation, for which the torque applied on the bacterial cell is much greater. Simulations performed with orientations in intermediate positions between the steamwise and wall-normal orientation confirm the trend of increasing torque magnitude at a larger angle with the flow field. As the torque is much greater in this orientation, a large angular acceleration is imposed, so the bacterium will reorient in a short time frame to the streamwise direction. This specific motion was observed for microorganisms in shear flows and is usually referred to as tumbling [95]. This non-uniform rotational motion will cause a probability distribution of bacterium orientations with a maximum close to the streamwise orientation. Such distribution was also experimentally observed for elongated cylinders in shear flows [94].

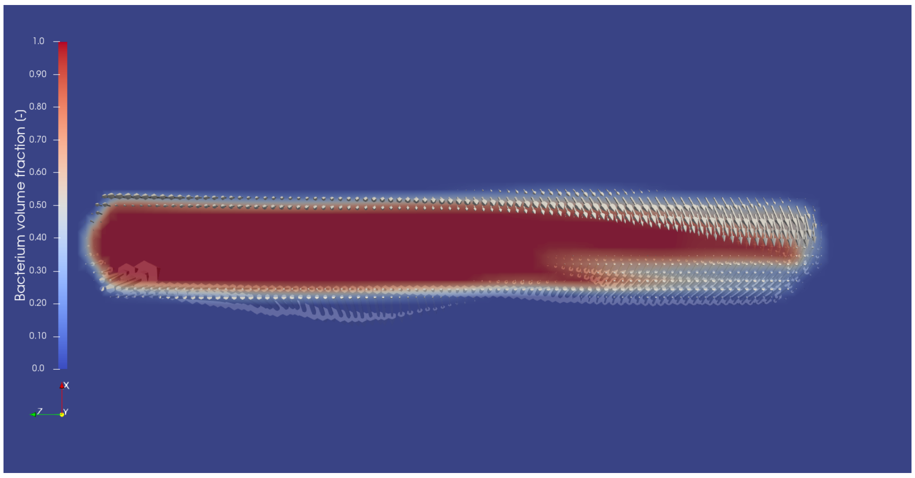
Finally, the stresses that result from the forces that act on the bacterial cell were determined. To study the possibility of plastic yielding of these bacteria, the Tresca criterion was used. This criterion predicts that failure of the material happens in shear, which is as expected in the turbulent Couette case, due to the high shear rates. When the maximum shear stress on any plane surpasses a critical value, yielding is expected to occur. The Tresca criterion gives the relation between the shear yield strength and the tensile yield strength, which was found to be that the shear yield strength is half of the tensile yield strength [96]. When this yield criterion is applied to the simulations, it was found that the yield limit is not surpassed for the lowest Reynolds numbers. At the highest simulated Reynolds number ($\mathrm{Re} = 1 \cdot 10^6$), it was found that the yield stress is surpassed by approximately 25 percent of the tensile yield strength and yielding is expected to occur in all bacterium orientations. When the Von Mises yield criterion is applied, which presumes that yielding occurs when the total strain energy density is larger than a critical value [96], same conclusions were obtained. To confirm that the yield stress is not surpassed in flows similar to the rs-SDR, a simulation in wall-normal orientation was performed with a Reynolds number of 825, which yields the same shear rate as the maximum shear rate in the rs-SDR used in this research at a rotational speed of 50 rad s^{-1}. A similar tumbling motion was simulated for several periods of rotation and the yield stress in the bacterial cell was not surpassed.



(a) Streamwise



(b) Wall-normal



(c) Spanwise

Figure 4.11: Drag forces acting on the bacterial cell for different orientation at a Reynolds number of 21333. The magnitude of the vectors are not to scale with respect to each individual figure.

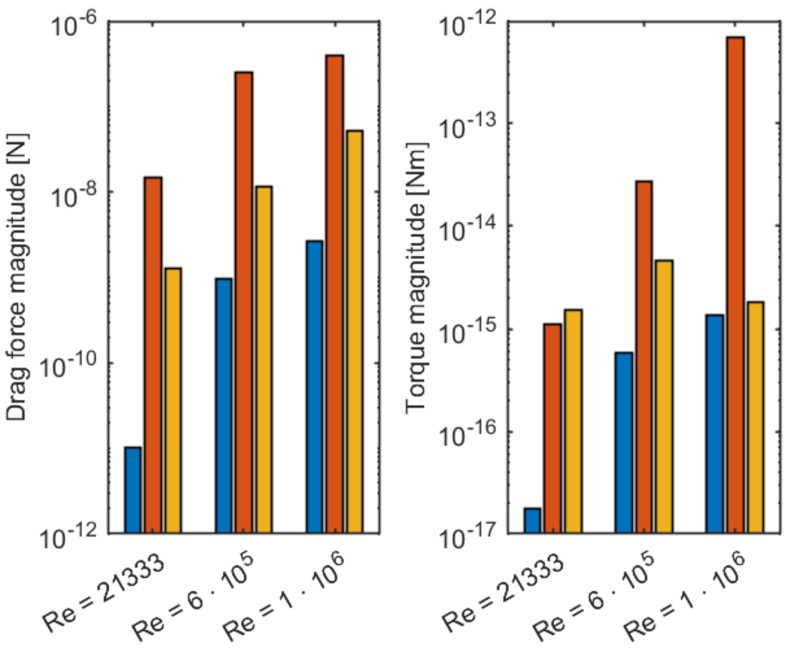


Figure 4.12: The magnitude of the drag force on the bacterium together with the resulting torque for different bacterial orientations and Reynolds numbers. The blue bars denote streamwise orientation of the bacterium, red denotes wall-normal orientation of the bacterium and yellow denotes a spanwise orientation.

4.4 Particle-laden flow in the rs-SDR

From the MP-PIC simulation in the rs-SDR, the instantaneous particle volume fraction distributions were studied. It was found that the particle dynamics are in line with the hydrodynamics as predicted from the CFD simulations in the rs-SDR. In the throughflow governed regime near the inlet, a uniform particle distribution was observed. In the rotation governed regime, the particles were found to recirculate in the Von Kármán and Bödewadt boundary layers. This yields initially high volume fractions of particles in the turbulent boundary layers in contrast to the inviscid core. After longer times, the particles propagate from the turbulent boundary layers to the inviscid core and a homogeneous particle distribution was observed in the rotation governed regime, similar to the throughflow governed regime. In Appendix M, the dynamics of the bacteria in the rs-SDR after injection is elaborated upon.

In Figure 4.13, the axial profiles of the particle volume fraction are given for different radial positions for the bottom and top gap. It can be seen that the particle distribution is flat throughout the axial gap between rotor and stator. For the particle distributions in the top gap and bottom gap, similar distributions were observed. Differences in the magnitude of the particle volume fraction between the top and bottom gap were found to be minimal. Effects of the centrifugal flow in the bottom gap and the centripetal flow in the top gap seem to have little influence on the particle volume fractions, as the density difference between the water and the bacteria is relatively low.

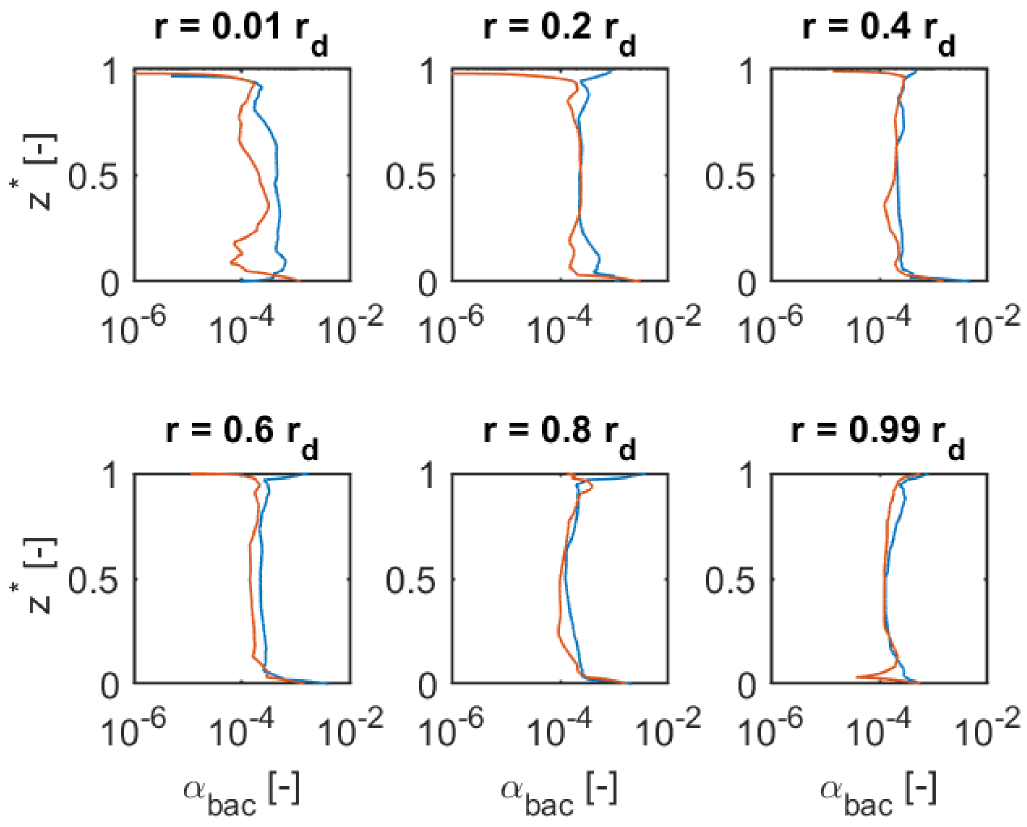


Figure 4.13: Particle volume fraction distributions in the rs-SDR at different radial positions in the lower gap (blue line) and top gap (red line).

Chapter 5
Conclusions

In this work, the hydrodynamics in the rs-SDR and its effect on bacteria confined in such flow were studied using CFD in OpenFOAM. Firstly, a validation case was set up on basis of a rotor-stator cavity as studied by Poncet et al. [57]. Using Large Eddy Simulations with the WALE sub-grid-scale turbulence model, satisfactory agreement with experimental result by Poncet et al. [57] was observed for the tangential velocity profiles. At high radial positions, the simulation was found to deviate significantly from the experimental results for the Reynolds stress profiles and radial velocity profiles.

The simulation of the cavity was successfully extended to describe the hydrodynamics in the rs-SDR with optional throughflow. A methodology was provided to construct the mesh that describes the rs-SDR and to numerically stabilise the simulations. Validation tests on the entrainment coefficient and boundary layer thickness were performed on the rs-SDR simulations and adequate agreement was observed. On basis of the radial velocity profiles, the transition between a throughflow governed regime and a rotation governed regime was observed. The transitional radius was estimated from these velocity profiles and it was observed to scale with the Reynolds number and throughflow coefficient as according to literature [57, 59]. Related to this, the RTDs were experimentally studied and compared with simulated RTDs. For the cases in the turbulent regime, the simulation was in agreement with the experiments, but the simulated RTD in the laminar regime underestimated the dispersion. To further study the macromixing in the rs-SDR, the engineering model was extended towards an axially-dispersed PFR and a CSTR in series. This proposed model described the breakthrough in the RTD better, but yields overall similar correspondence with the engineering model with a non-integer number of tanks.

A high-Reynolds number Couette flow was simulated and validated to be used as a flow field to simulate mechanical shear effects on the bacterial cell. Using an Eulerian simulation with the Volume of Fluid method, the forces on the bacterial scale related to the shear effects were simulated. The drag force acting on the bacterium was found to be the dominant force, which causes a rotation in the shear plane. From these forces, a tumbling motion of the bacterial cell was predicted with a preferential alignment in the flow direction. Using the Tresca and Von Mises failure criteria, plastic deformation at the shear rates in the rs-SDR was not predicted for the timespan measured.

Finally, the possibility for simulating particle-laden flow in the rs-SDR using an Eulerian-Lagrangian simulation with the multiphase particle-in-cell model was explored. Such simulation was constructed to describe the bacteria-laden flow and particle dynamics were observed to be similar to the overall hydrodynamics in the rs-SDR, with uniform flow in the throughflow governed zone and recirculation in the Bödewadt and Von Kármán boundary layers in the rotation governed zone. As the local shear rates are higher in these turbulent boundary layers, this emphasizes the need for studying the (long-term) shear effects on the bacteria when operated in a rs-SDR.

Chapter 6 Recommendation

In this work, a methodology was provided to simulate the turbulent flow in the rotor-stator spinning disc reactor. The verification tests that were employed to assess the quality of the simulation results are satisfactory, but additional verification and validation would improve the quality of the research. One of such would be to compare experimentally obtained micromixing times with the micromixing time as predicted from the CFD simulations. The axial profiles of the micromixing times currently do not correspond to experimentally obtained micromixing times, but it must be noted that these experiments are intrusive, which can distort the local hydrodynamics.

Additionally, a validation technique should be implemented to validate the particle dynamics in the rs-SDR. Especially due to the non-spherical geometry of the bacteria, the utilised force correlations are strictly speaking not valid. Additionally, the great amount of input parameters makes validation important for the implementation of the MP-PIC model. Therefore, a non-invasive validation technique is required to validate the particle dynamics. One possibility to do so is by making use of magnetic particle tracking, which tracks a single particle over time through the measurement of a generated magnetic field [97, 98]. Another technique to study the particle dynamics would be optical methods, but since these rely on continuous visibility of the particles, the geometry remains limited to 2D cases [98], making it less suitable for the rs-SDR.

To study the mechanical effects on the bacterial cell during confinement in shear flows, a simplified simulation was constructed that considers the bacterial cell to be a pseudofluid with a high viscosity. In order to improve the simulations of bacterial cells in shear flows, more experimental research is required on the bacterium *Clostridium autoethanogenum*. Currently, the mechanical properties of the bacterial cell were determined from viscoelastic measurements on the bacterial thread made of peptidoglycan. However, the bacterial cell wall does not exclusively consist of peptidoglycan and is most likely anisotropic in mechanical properties. Additionally, it is known that bacteria can move in various direction due to chemotaxis, gyrotaxis or gravitaxis [99, 100]. These are topics which are still not researched for *Clostridium autoethanogenum* and should be studied before expanding the current simulation. To take into account the anisotropy of the bacterial cell, a spring-based model could be implemented [48], which would require development of an inhouse code.

It has been shown that CFD is a valuable tool to predict the hydrodynamics in the rs-SDR with feasible simulation times. Extensions of the simulations to describe multiphase and/or reactive systems can be important for further optimization and development of process intensification in the rs-SDR.

Chapter 7
Acknowledgements

First of all, I would like to thank John van der Schaaf for his input during the progress meetings. He gave me many new insights on other details that could be studied using the simulations that I performed and boosted my energy to dive back into the code after each meeting. I enjoyed being a part of SPE.

Next to this, I would like to thank my daily supervisor Vince Hop. I am thankful for the challenging project and his expertise on modelling aided me very much. I enjoyed the informal chats that we had during my daily collection of the simulation results. Without all his efforts in submitting the simulations to the supercomputer at any time, the project would have proceeded much slower. I also appreciated the amount of trust Vince had in me and how he treated me as a peer in tackling the problems we encountered, which made me get the best out of me.

Additionally, I would like to express my gratitude to Arnab Chaudhuri for his help with the experimental set-up and his valuable input during the project.

Finally, I would like to show my appreciation to my office mates in ’de Broeikas’, Aysima, Boris, Charlotte, Floris, Joost, Jordy, Koen, Lars, Lieke, Marieke, Robin, Sare, Senan and Zafina. With all the laughter, coffee breaks and our shared struggles, you really helped me enjoy my time at the research group.

This work made use of the Dutch national e-infrastructure with the support of the SURF Cooperative using grant no. EINF-2666

Bibliography

(1) Marusic, I.; Broomhall, S. Leonardo da Vinci and Fluid Mechanics, 2021.

(2) Bilgen, S. *Renewable and Sustainable Energy Reviews* **2014**, *38*, 890–902.

(3) Abdelaziz, E. A.; Saidur, R.; Mekhilef, S. A review on energy saving strategies in industrial sector, 2011.

(4) Yung, M. M.; Jablonski, W. S.; Magrini-Bair, K. A. *Energy and Fuels* **2009**, *23*, 1874–1887.

(5) Pan, S. Y.; Chiang, P. C.; Pan, W.; Kim, H. *Critical Reviews in Environmental Science and Technology* **2018**, *48*, 471–534.

(6) Subramani, V.; Gangwal, S. K. *Energy and Fuels* **2008**, *22*, 814–839.

(7) Jadhav, S. G.; Vaidya, P. D.; Bhanage, B. M.; Joshi, J. B. *Chemical Engineering Research and Design* **2014**, *92*, 2557–2567.

(8) Panzone, C.; Philippe, R.; Chappaz, A.; Fongarland, P.; Bengaouer, A. *Journal of CO2 Utilization* **2020**, *38*, 314–347.

(9) Salehizadeh, H.; Yan, N.; Farnood, R. *Chemical Engineering Journal* **2020**, *390*, 124584.

(10) Chen, M.; Liu, M.; Tang, Y. *International Journal of Chemical Reactor Engineering* **2019**, *17*, DOI: `10.1515/ijcre-2018-0254`.

(11) Irfan, M.; Bai, Y.; Zhou, L.; Kazmi, M.; Yuan, S.; Maurice Mbadinga, S.; Yang, S. Z.; Liu, J. F.; Sand, W.; Gu, J. D.; Mu, B. Z. *Bioresource Technology* **2019**, *288*, 121401.

(12) Mohammadi, M.; Najafpour, G. D.; Younesi, H.; Lahijani, P.; Uzir, M. H.; Mohamed, A. R. *Renewable and Sustainable Energy Reviews* **2011**, *15*, 4255–4273.

(13) Abubackar, H. N.; Veiga, M. C.; Kennes, C. *Bioresource Technology* **2015**, *186*, 122–127.

(14) Asimakopoulos, K.; Gavala, H. N.; Skiadas, I. V. *Chemical Engineering Journal* **2018**, *348*, 732–744.

(15) Meeuwse, M.; van der Schaaf, J.; Kuster, B. F.; Schouten, J. C. *Chemical Engineering Science* **2010**, *65*, 466–471.

(16) Meeuwse, M.; Hamming, E.; van der Schaaf, J.; Schouten, J. C. *Chemical Engineering and Processing: Process Intensification* **2011**, *50*, 1095–1107.

(17) Meeuwse, M.; van der Schaaf, J.; Schouten, J. C. *AIChE Journal* **2012**, *58*, 247–255.

(18) De Beer, M. M.; Keurentjes, J. T.; Schouten, J. C.; Van der Schaaf, J. *Chemical Engineering Journal* **2014**, *242*, 53–61.

(19) de Beer, M. Hydrodynamics and heat transfer of single and multiphase flows in rotor-stator spinning disc reactors, Ph.D. Thesis, Chemical Engineering and Chemistry, 2016.

(20) Davidson, P., *Turbulence: an introduction for scientists and engineers*, Second Edition; Oxford University Press.: Oxford, UK: 2005.

(21) Versteeg, H. K.; Malalasekera, W., *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*; Pearson Education Limited: 2007.

(22) Sprott, J. C., *Chaos and Time-Series Analysis*; Oxford University Press, Inc.: USA, 2003.

(23) Yamada, M.; Ohkitani, K. *Physical Review Letters* **1988**, *60*, 983–986.

(24) Kalmár-Nagy, T.; Bak, B. D. *Nonlinear Dynamics* **2019**, *95*, 3193–3203.

(25) Kolmogorov, A. N. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* **1991**, *434*, 9–13.

(26) Kolmogorov, A. N. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* **1991**, *434*, 15–17.

(27) Lesieur, M., *Turbulence in fluids.* Fluid Mechanics and its Applications, Vol. 84; Springer Netherlands: Dordrecht, 1987.

(28) Mason, P. J. *Quarterly Journal of the Royal Meteorological Society* **1994**, *120*, 1–26.

(29) Smagorinsky, J. *Monthly Weather Review* **1963**, *91*, 99–164.

(30) Lilly, D. K. *IBM Form* **1967**, 195–210.

(31) Lilly, D. K. *NCAR Manuscript* **1966**, *123*.

(32) Nicoud, F.; Ducros, F. *Flow, Turbulence and Combustion* **1999**, *62*, 183–200.

(33) Ii, S.; Sugiyama, K.; Takagi, S.; Matsumoto, Y. *Journal of Biomechanical Science and Engineering* **2012**, *7*, 72–83.

(34) Jafari, A.; Mousavi, S. M.; Kolari, P. *Communications in Nonlinear Science and Numerical Simulation* **2008**, *13*, 1615–1626.

(35) Jafari, A.; Zamankhan, P.; Mousavi, S. M.; Kolari, P. *Communications in Nonlinear Science and Numerical Simulation* **2009**, *14*, 1396–1402.

(36) Balachandran Nair, A. N.; Pirker, S.; Saeedipour, M. *Computational Particle Mechanics* **2021**, DOI: `10.1007/s40571-021-00441-x`.

(37) Zhang, Z.; Xu, J.; Chen, X. In *Modeling Cell Deformation in CTC Microfluidic Filters*, 2014; Vol. 3.

(38) Ariyaratne, W. H.; Manjula, E.; Ratnayake, C.; Melaaen, M. C. *Proceedings of The 9th EUROSIM Congress on Modelling and Simulation, EUROSIM 2016, The 57th SIMS Conference on Simulation and Modelling SIMS 2016* **2018**, *142*, 680–686.

(39) Wardle, K. E.; Weller, H. G. *International Journal of Chemical Engineering* **2013**, *2013*, DOI: `10.1155/2013/128936`.

(40) Brackbill, J. U.; Kothe, D. B.; Zemach, C. *Journal of Computational Physics* **1992**, *100*, 335–354.

(41) Hirt, C. W.; Nichols, B. D. *Journal of Computational Physics* **1981**, *39*, 201–225.

(42) Van Sint Annaland, M.; Deen, N. G.; Kuipers, J. A. *Chemical Engineering Science* **2005**, *60*, 2999–3011.

(43) Roohi, E.; Zahiri, A. P.; Passandideh-Fard, M. *Applied Mathematical Modelling* **2013**, *37*, 6469–6488.

(44) Hoomans, B. Granular dynamics of gas-solid two-phase flows, English, Ph.D. Thesis, University of Twente, 2000.

(45) Kuruneru, S. T.; Sauret, E.; Saha, S. C.; Gu, Y. T. *International Communications in Heat and Mass Transfer* **2017**, *89*, 176–184.

(46) Elghobashi, S. *Applied Scientific Research* **1994**, *52*, 309–329.

(47) Greifzu, F.; Kratzsch, C.; Forgber, T.; Lindner, F.; Schwarze, R. *Engineering Applications of Computational Fluid Mechanics* **2016**, *10*, 30–43.

(48) Ju, M.; Ye, S. S.; Namgung, B.; Cho, S.; Low, H. T.; Leo, H. L.; Kim, S. *Computer Methods in Biomechanics and Biomedical Engineering* **2015**, *18*, 130–140.

(49) Secomb, T. W.; Styp-Rekowska, B.; Pries, A. R. *Annals of Biomedical Engineering* **2007**, *35*, 755–765.

(50) Jung, J.; Hassanein, A. *Medical Engineering and Physics* **2008**, *30*, 91–103.

(51) Daily, J. W.; Nece, R. E. *Journal of Fluids Engineering, Transactions of the ASME* **1960**, *82*, 217–230.

(52) Launder, B.; Poncet, S.; Serre, E. *Annual Review of Fluid Mechanics* **2010**, *42*, 229–248.

(53) Kármán, T. V. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik* **1921**, *1*, 233–252.

(54) Bödewadt, U. T. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik* **1940**, *20*, 241–253.

(55) Owen, J.; Rogers, R., *Flow and Heat Transfer in Rotating Disc Systems, Vol.1: Rotor-Stator Systems*; Research Studies Press: 1989.

(56) Poncet, S.; Schiestel, R.; Chauve, M. P. *Journal of Fluids Engineering, Transactions of the ASME* **2005**, *127*, 787–794.

(57) Poncet, S.; Chauve, M. P.; Schiestel, R. *Physics of Fluids* **2005**, *17*, 1–15.

(58) Haddadi, S.; Poncet, S. *International Journal of Rotating Machinery* **2008**, *2008*, DOI: 10.1155/2008/635138.

(59) Phadke, U. P.; Owen, J. M. *International Journal of Heat and Fluid Flow* **1988**, *9*, 98–105.

(60) Nguyen, N. D.; Ribault, J. P.; Florent, P. *Journal of Fluid Mechanics* **1975**, *68*, 369–388.

(61) Abrini, J.; Naveau, H.; Nyns, E. J. *Archives of Microbiology* **1994**, *161*, 345–351.

(62) Abubackar, H. N.; Bengelsdorf, F. R.; Dürre, P.; Veiga, M. C.; Kennes, C. *Applied Energy* **2016**, *169*, 210–217.

(63) Bratbak, G.; Dundas, I. *Applied and Environmental Microbiology* **1984**, *48*, 755–757.

(64) Coelho, F. M.; Nele, M.; Ribeiro, R. R.; Ferreira, T. F.; Amaral, P. F. *Chemical Engineering Transactions* **2016**, *50*, 277–282.

(65) Kim, S. J.; Chang, J.; Singh, M. *Biochimica et Biophysica Acta (BBA) - Biomembranes* **2015**, *1848*, 350–362.

(66) Mendelson, N. H.; Thwaites, J. J. *Journal of bacteriology* **1989**, *171*, 1055–1062.

(67) Thwaites, J. J.; Mendelson, N. H. *International Journal of Biological Macromolecules* **1989**, *11*, 201–206.

(68) Thwaites, J. J.; Mendelson, N. H. *Advances in Microbial Physiology* **1991**, *32*, 173–222.

(69) Chen, G.; Xiong, Q.; Morris, P. J.; Paterson, E. G.; Sergeev, A.; Wang, Y. *Not. AMS* **2014**, *61*, 354–363.

(70) Greenshields, C. J., *User Guide, Version 9*; OpenFOAM Foundation Ltd.: 2021.

(71) Zirwes, T.; Zhang, F.; Habisreuther, P.; Hansinger, M.; Bockhorn, H.; Pfitzner, M.; Trimis, D. *Flow, Turbulence and Combustion* **2020**, *104*, 997–1027.

(72) Feng, J.; Chen, H.; He, Q.; Ye, M. *Fusion Engineering and Design* **2015**, *100*, 260–264.

(73) Van Leer, B. *Journal of Computational Physics* **1974**, *14*, 361–370.

(74) Liu, Y.; Hinrichsen, O. *Computers & Chemical Engineering* **2014**, *69*, 75–88.

(75) Vreugdenhil, C. A.; Taylor, J. R. *Physics of Fluids* **2018**, *30*, DOI: 10.1063/1.5037039.

(76) Séverac, É.; Poncet, S.; Serre, É.; Chauve, M. P. *Physics of Fluids* **2007**, *19*, DOI: 10.1063/1.2759530.

(77) Šidlof, P.; Horáček, J.; Řidký, V. *Computers & Fluids* **2013**, *80*, 290–300.

(78) Wang, G.; Yang, F.; Wu, K.; Ma, Y.; Peng, C.; Liu, T.; Wang, L. P. *Chemical Engineering Science* **2021**, *229*, DOI: 10.1016/j.ces.2020.116133.

(79) Roos Launchbury, D., *Unsteady Turbulent Flow Modelling and Applications*; Springer Fachmedien Wiesbaden: 2016.

(80) Burgos, G. R.; Alexandrou, A. N.; Entov, V. *Journal of Rheology* **1999**, *43*, 463–483.

(81) Kumar, M.; Reddy, R.; Banerjee, R.; Mangadoddy, N. *Separation and Purification Technology* **2021**, *266*, 118206.

(82) Andrews, M. J.; O'Rourke, P. J. *International Journal of Multiphase Flow* **1996**, *22*, 379–402.

(83) Snider, D. M. *Journal of Computational Physics* **2001**, *170*, 523–549.

(84) Clark, S.; Snider, D. M.; Spenik, J. *Powder Technology* **2013**, *242*, 100–107.

(85) Jang, K.; Han, W.; Huh, K. Y. In *OpenFOAM® : Selected Papers of the 11th Workshop*, Nóbrega, J. M., Jasak, H., Eds.; Springer International Publishing: Cham, 2019, pp 419–435.

(86) Haider, A.; Levenspiel, O. *Powder Technology* **1989**, *58*, 63–70.

(87) Lataste, J.; Huilier, D.; Burnage, H.; Bednář, J. *Atmospheric Environment* **2000**, *34*, 3963–3971.

(88) Freeman, J. O.; Peterson, S.; Cao, C.; Wang, Y.; Franklin, S. V.; Weeks, E. R. *Granular Matter* **2019**, *21*, 1–11.

(89) Chaudhuri, A. Intensification of Multiphase Systems with Rotor-Stator Spinning Disk Reactors, Ph.D. Thesis, Chemical Engineering and Chemistry, 2021.

(90) Van Eeten, K. M.; van der Schaaf, J.; Schouten, J. C.; van Heijst, G. J. *Physics of Fluids* **2012**, *24*, DOI: 10.1063/1.3698406.

(91) Pirozzoli, S.; Bernardini, M.; Orlandi, P. *Journal of Fluid Mechanics* **2014**, *758*, 327–343.

(92) Trevelyan, B. J.; Mason, S. G. *Journal of Colloid Science* **1951**, *6*, 354–367.

(93) Zettner, C. M.; Yoda, M. *Journal of Fluid Mechanics* **2001**, *442*, 241–266.

(94) Wegner, S.; Börzsönyi, T.; Bien, T.; Rose, G.; Stannarius, R. *Soft Matter* **2012**, *8*, 10950–10958.

(95) Pedley, T. J.; Kessler, J. O. *Proceedings of the Royal Society of London - Biological Sciences* **1987**, *230*, 47–70.

(96) Jones, R. M., *Deformation theory of plasticity*; Bull ridge Publishing: Blacksburg, Virginia, 2009, pp 146–160.

(97) Buist, K. A.; van der Gaag, A. C.; Deen, N. G.; Kuipers, J. A. *AIChE Journal* **2014**, *60*, 3133–3142.

(98) Nijssen, T. M.; Hoeks, I.; Manjunath, V.; Kuipers, H. A.; van der Stel, J.; Adema, A. T.; Buist, K. A. *Chemical Engineering Science: X* **2022**, *13*, 100120.

(99) Kessler, J. O. *Nature* **1985**, *313*, 218–220.

(100) Wadhams, G. H.; Armitage, J. P. Making sense of it all: Bacterial chemotaxis, 2004.

(101) Telbany, M. M.; Reynolds, A. J. *Journal of Fluid Mechanics* **1980**, *100*, 1–29.

(102) Kitoh, O.; Nakabyashi, K.; Nishimura, F. *Journal of Fluid Mechanics* **2005**, *539*, 199–227.

(103) Dinh, V. P.; Huynh, T. D. T.; Le, H. M.; Nguyen, V. D.; Dao, V. A.; Hung, N. Q.; Tuyen, L. A.; Lee, S.; Yi, J.; Nguyen, T. D.; Tan, L. V. *RSC Advances* **2019**, *9*, 25847–25860.

(104) Fogler, H. S., *Elements of Chemical Reaction Engineering (4th Edition)*, 4th ed.; Prentice Hall: 2005.

(105) Bothe, D.; Schmidtke, M.; Warnecke, H. J. *Chemical Engineering and Technology* **2006**, *29*, 1048–1053.

Appendix A Grid dependency study Couette flow

	N_x	N_y	N_z
Coarse	40	85	40
Middle	50	110	50
Fine	65	140	65
Unresolved	50	50	50

Table A.1: The number of grid cells used in the grid dependency study of the turbulent Couette flow

In Figure A.1, the grid dependency study for a turbulent Couette flow with a Reynolds number equal to 21333 is given. In this figure, several simulations are compared with DNS data as found by Pirozzoli et al. [91]. The number of grid cells in each direction is given in Table A.1. The methodology for these simulations was discussed in Section 3.3.1. It can be seen that all simulations are relatively close to the DNS results. The simulation without wall-resolving closely matches the coarse mesh, apart from the locations close to the walls. Due to the wall function that was forced at these locations, a different friction velocity was found. The coarse mesh was found to be unable to accurately describe the friction velocity near the wall, which is related to a y^+ value which was too large to resolve the viscous sublayer. The middle mesh did not suffer from these problems and the friction velocity was accurately resolved. The fine mesh did not show significant differences in the velocity profile. To limit the computational time while resolving the viscous sublayer, the middle mesh was selected in further simulations. For the bacterial deformation simulations, a finer grid was used compared to the middle mesh. This mesh was selected on basis of the spatial domain of the bacterial deformation simulation to ensure similar non-dimensional grid cell sizes. In the spanwise direction, a finer mesh was used to resolve the bacterial cell. Additionally, it must be noted that acquiring a grid-independent solution is infeasible for LES simulations, since the grid size directly determines the cutoff width for the eddies that are resolved.

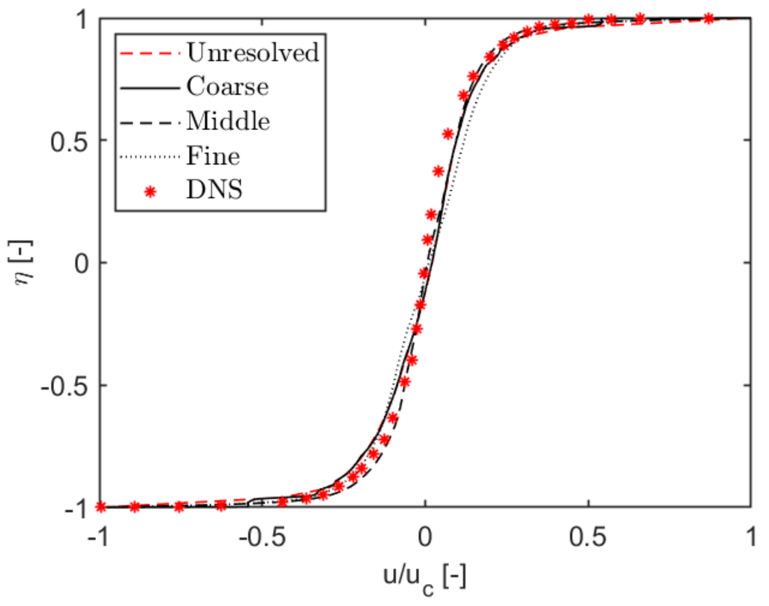


Figure A.1: The grid dependency study for a turbulent Couette flow with Re = 21333. The DNS data was reproduced from Pirozzoli et al. [91]. The unresolved case denotes a mesh without wall-resolving.

Appendix B
Results turbulent Couette flow

In this Appendix, the results of the turbulent Couette velocity profiles are discussed, that were used as an initial field in the bacterial-resolved flow simulations. In Figure B.1, the streamwise velocity profiles are depicted for the simulated Reynolds numbers. For the simulation at a Reynolds number of 21333, a RAS simulation using the $k - \omega$ shear stress transport model was performed as a verification case. It can be seen that the RAS simulation predicts a steeper velocity profile in the core, which is associated to the higher turbulence intensity predicted by RAS. The near-wall behaviour of both simulations shows similar characteristics, with the characteristic slip zones near the walls. It can be seen that with increasing Reynolds number, the average inner-scaled velocity slope ($S = (h/u_c)(d\overline{u}/dy)$) decreases. This effect was also reported by Pirozolli et al. [91], although a quantitative comparison of this trend with experimental results from literature yields different trends [101, 102]. This discrepancy is most likely related to experimental difficulty measuring relatively small velocity gradients in the core of the flow.

It can be seen that compared with the low Reynolds number simulation, the higher Reynolds number simulation show some oscillatory behaviour in the core of the flow. This is related to large-scale motions that form in the channel core. In the spanwise direction, oscillating streamwise velocity zones are observed, as depicted in Figure B.2. Pirozolli et al. [91] studied the spanwise two-point correlation function of the streamwise velocity and found results demonstrating the sinusoidal behaviour. With increasing Reynolds number, the confinement effects showed to increase. A similar trend was found in this study, as depicted in Figure B.2. Possible remedies to reduce the effect of these large-scale motions are spanwise averaging or averaging on a timescale equal to the period of such wave. However, since the sinusoidal behaviour of the large-scale motion in the spanwise direction yields a supposedly infinite correlation length, simulations and experiments of Couette flows are likely to be contaminated by confinement effects [91], which highlights the complexity of this type of flow.

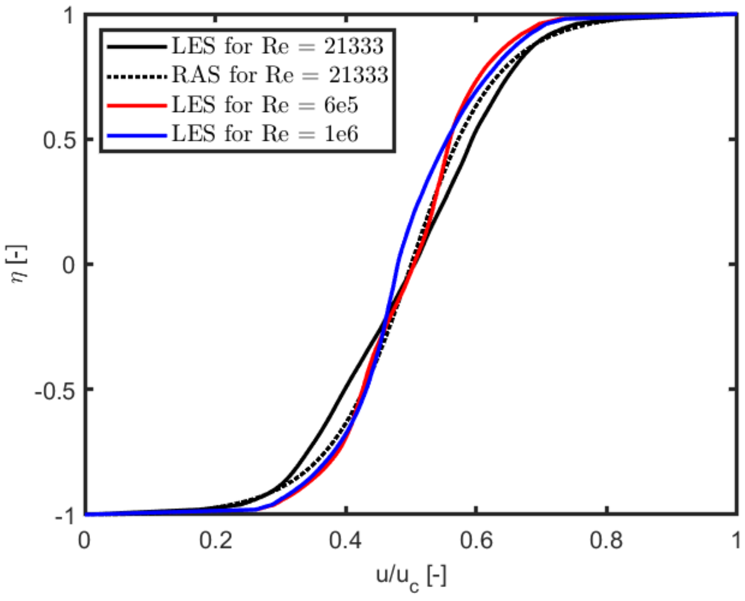


Figure B.1: Results of the Couette cases that are to be used as an initial field in the bacterial-resolved flow simulations. As a verification, a simulation with RAS was included.

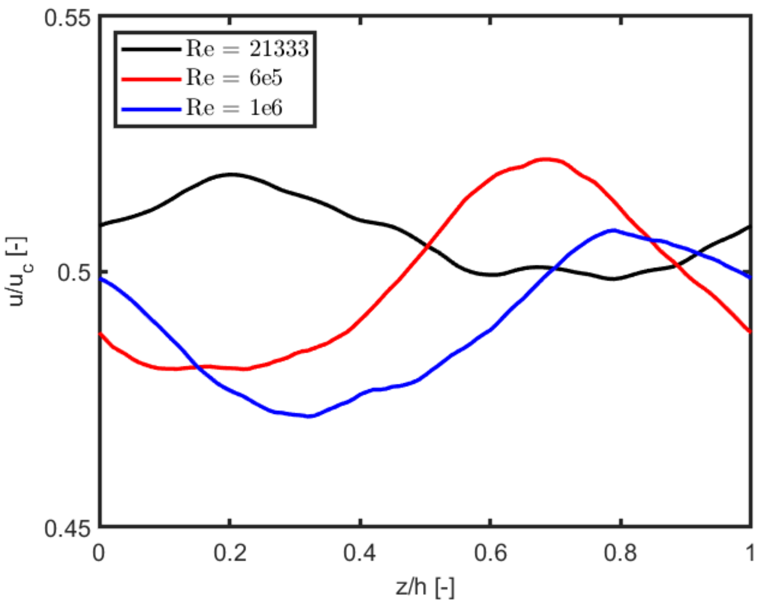


Figure B.2: Spanwise profiles of the streamwise velocity at the symmetry plane ($\eta = 0$) showing the large-scale motions due to confinement effects.

Appendix C

Geometry of bacterial cell

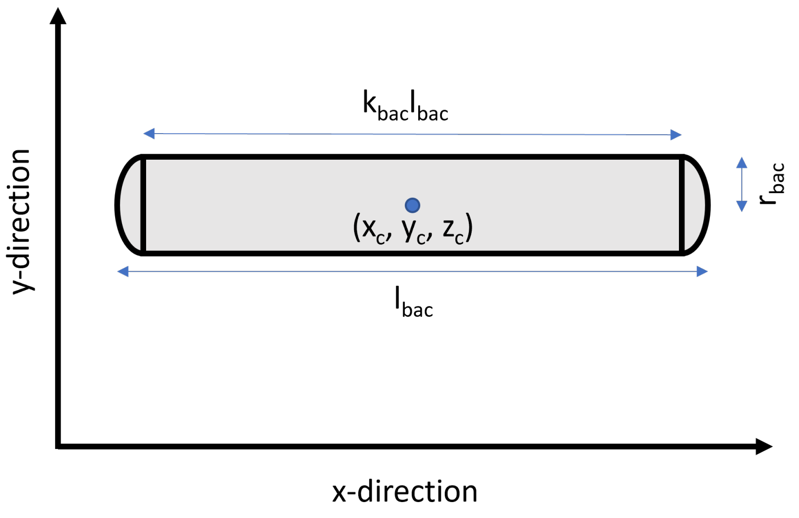


Figure C.1: Schematic overview of the initialisation of the bacterial cell in the flow field.

The geometry in which the bacterial cell was initialised in the simulation is discussed in this section. In Figure C.1, a schematic representation of the bacterial cell is provided. The bacterial cell consists of a cylinder with two ellipsoids at the caps of the cylinder. Using microscope images of the bacterial cells (see Figure C.2), it was established that the fraction of the bacterial cell that consists of the straight centre (k_{bac}) is approximately 0.9. This means that the relative size of the ellipsoid caps is approximately 0.05 for each cap, respectively. The bacterium is initialised in the center of the Couette flow, but the initialisation is flexible such that the bacterial cell can be initialised at any location (x_c, y_c, z_c). The cylinder can be described by Equation C.1, for the case in which the bacterial cell is oriented in the x-direction. The domain of the bacterial cell (the x-coordinate in Equation C.1) is related to the length of the bacterium. The ellipsoids that represent the caps of the bacterial cell can be described by Equations C.2 and C.3.

$$(y - y_c)^2 + (z - z_c)^2 \leq r_{bac}^2 \quad \wedge \quad x_{begin} \leq x \leq x_{end} \tag{C.1}$$

$$\frac{(x - (x_c - 0.5k_{bac}l_{bac}))^2}{((1 - k_{bac})l_{bac})^2} + \frac{(y - y_c)^2}{r_{bac}^2} + \frac{(z - z_c)^2}{r_{bac}^2} \leq 1 \tag{C.2}$$

$$\frac{(x - (x_c + 0.5k_{bac}l_{bac}))^2}{((1 - k_{bac})l_{bac})^2} + \frac{(y - y_c)^2}{r_{bac}^2} + \frac{(z - z_c)^2}{r_{bac}^2} \leq 1 \tag{C.3}$$

To study the effect of the orientation of the bacterial cell on the local hydrodynamics and forces acting on the bacterium, the initial situation was rotated around the z-axis with an angle θ. This rotation corresponds to a matrix operation as defined in Equation C.4. This results in the coordinate transformation as depicted in Equations C.5 to C.7, which can be applied to the equations that describe the geometry of the bacterium. In these equations, the translations are included that ensure a rotation around the center point (x_c, y_c, z_c).

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{C.4}$$

$$x' = (x - x_c)\cos\theta - (y - y_c)\sin\theta + x_c \tag{C.5}$$
$$y' = (x - x_c)\sin\theta + (y - y_c)\cos\theta + y_c \tag{C.6}$$
$$z' = z \tag{C.7}$$

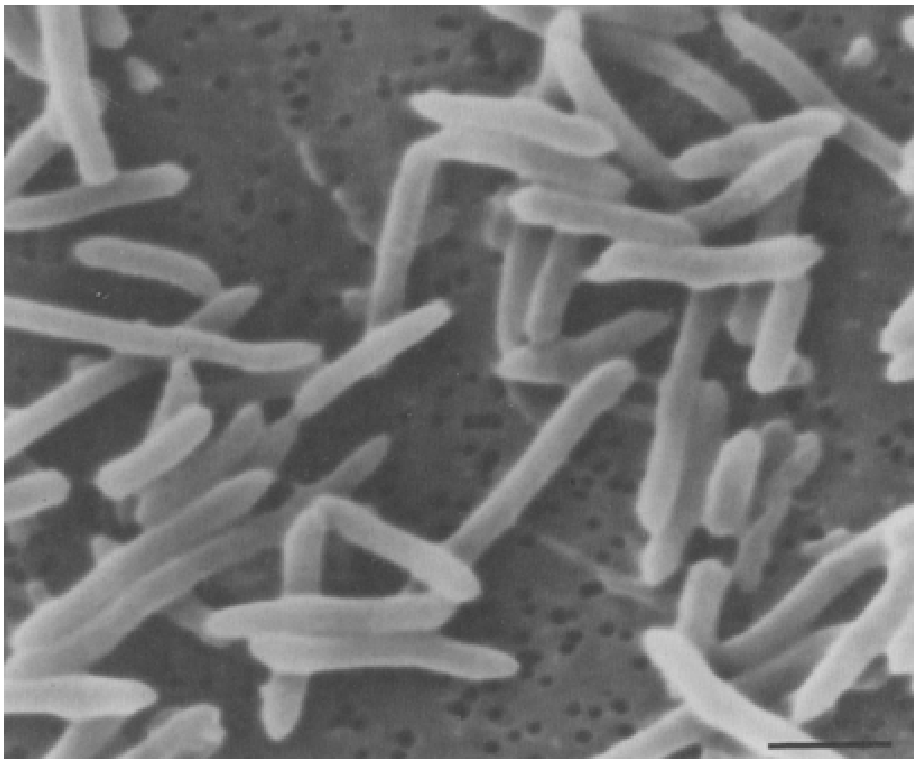
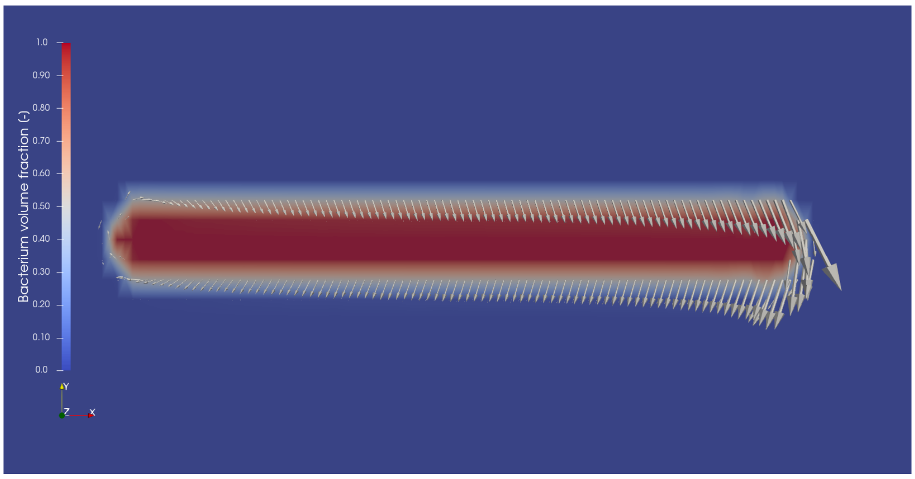


Figure C.2: Scanning electron microscopy picture of *Clostridium autoethanogenum*. Reproduced from [61].

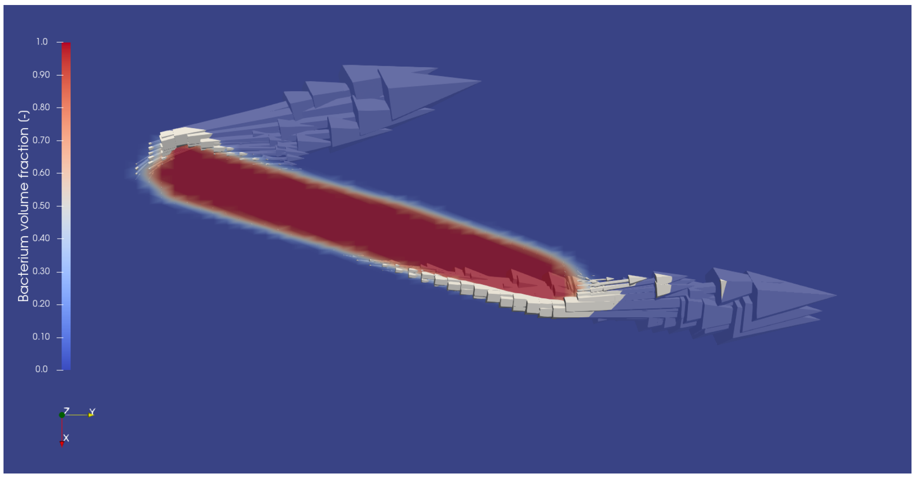
Appendix D
Results of bacterial-resolved flow

In Figures D.1 and D.2, the orientation of the drag force on the bacterial cell is visualised for the other two simulated Reynolds numbers and bacterium orientations. It must be noted that the magnitude of the vectors is not to scale in comparison with the other figures. In this figure, the x-coordinate denotes the streamwise direction, the y-coordinate denotes the wall-normal direction and the z-coordinate denotes the spanwise direction. Similar trends as discussed in Section 4.3 can be observed. Next to this, it can be observed that deformation and yielding occurs when the bacterium is oriented in the wall-normal direction at a Reynolds number of $1 \cdot 10^6$ (Figure D.2b). This is in correspondence with the predictions using the Von Mises and Tresca criterion and the simulated shear stresses in the bacterial cell. Due to the available computational resources, simulation of one complete large-scale eddy-turnover time was unfeasible. For each simulation, the normalised simulation time by the eddy turn-over time is approximately 0.2. The eddy turn-over time, as defined in Equation D.1, can be described as the characteristic time scale at which one large-scale eddy is dissipated into heat according to the energy cascade. Therefore, the influence of large-scale eddies on the bacterial cell is not averaged out sufficiently and the results should be interpreted with this caveat in mind.

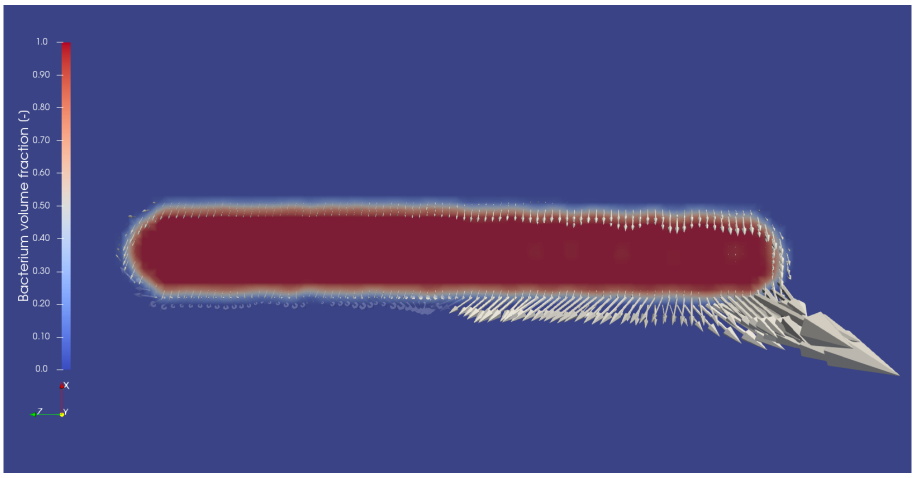
$$t_{eddy} = \frac{h}{u_{wall}} \tag{D.1}$$



(a) Streamwise

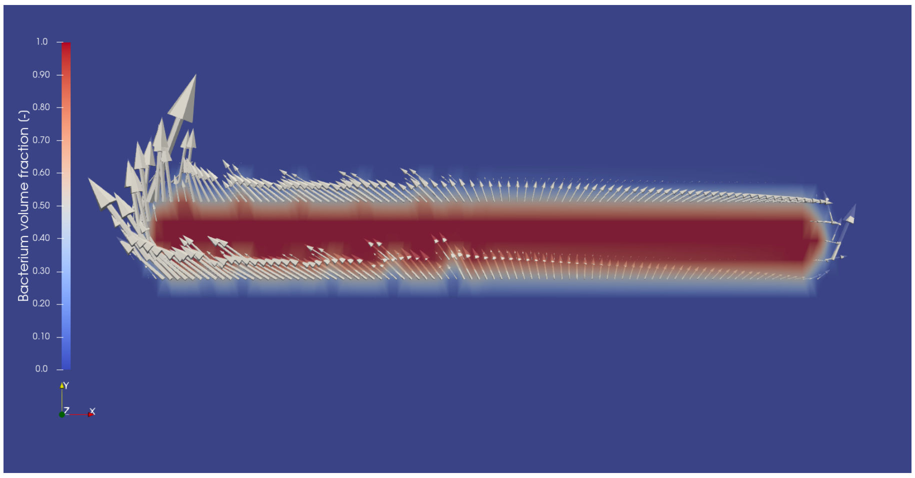


(b) Wall-normal

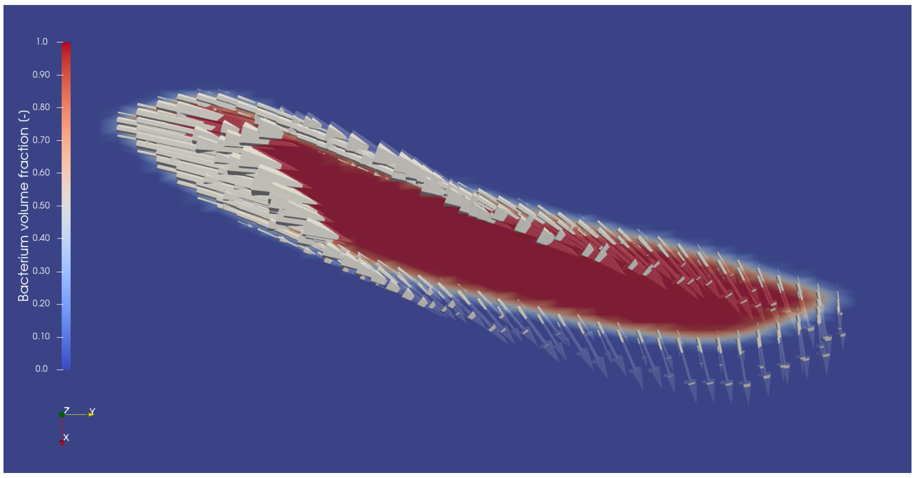


(c) Spanwise

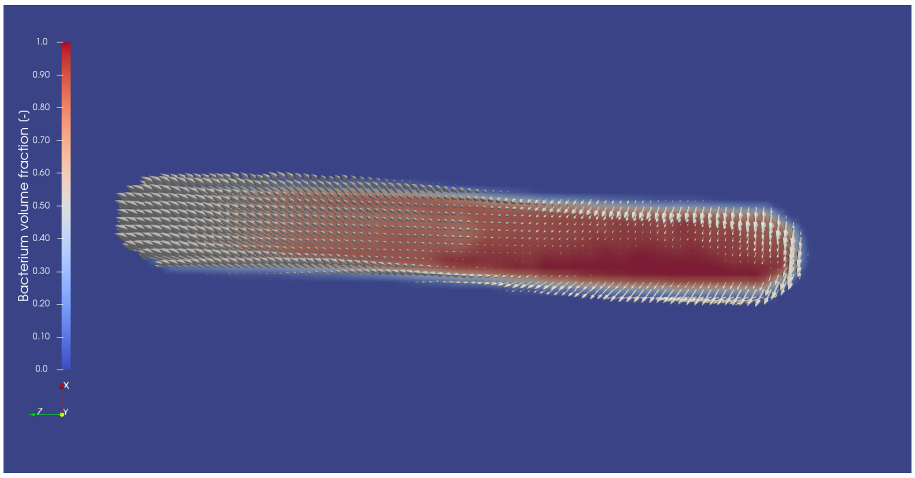
Figure D.1: Drag forces acting on the bacterial cell for different orientation at a Reynolds number of $6 \cdot 10^5$. The magnitude of the vectors are not to scale with respect to each individual figure.



(a) Streamwise



(b) Wall-normal



(c) Spanwise

Figure D.2: Drag forces acting on the bacterial cell for different orientation at a Reynolds number of $1 \cdot 10^6$. The magnitude of the vectors are not to scale with respect to each individual figure.

Appendix E
Scalability of parallel performance

In Figure E.1, the time required for one iteration of the simulation of a rotor-stator cavity is plotted as a function of the number of cores used in the parallel computation. This graph was obtained by simulating this case for 100 time steps. The last four time steps before the simulation would be finalised were taken into consideration in the production of this graph. Of these four time steps, the mean time per iteration was computed and is depicted in Figure E.1. This was done for the two super computers that were used in this research, Cartesius and Snellius. It can be seen that increasing the number of cores is beneficial for the computational time, but increasing the number of cores further is not expected to increase the computational speed. Additionally, it can be noted that the upgrade from Cartesius to Snellius decreased the computational time slightly. As a reference, the ideal time per iteration is given, which was computed by assuming that the computational time is halved with a double number of cores.

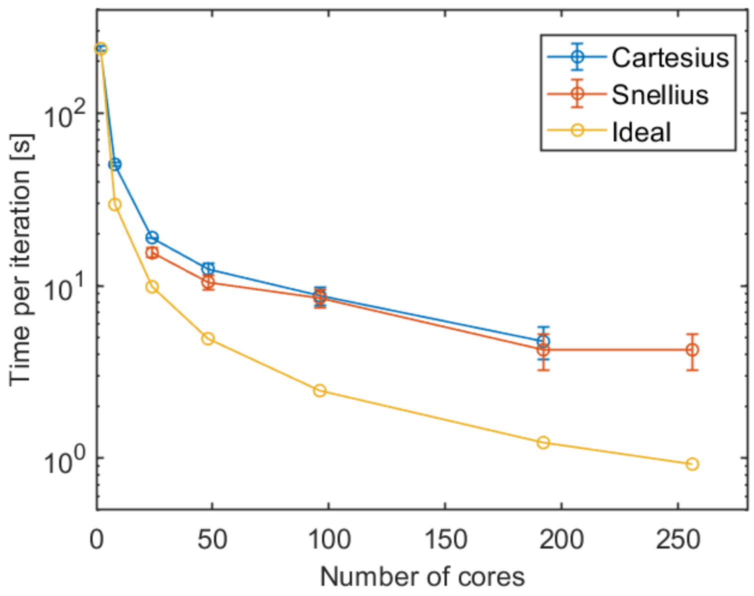


Figure E.1: The time per iteration in the rotor-stator cavity simulation as a function of the number of cores that were used in parallel computations. The ideal scalability is included as reference. The error bars denote the maximum deviation between the observed iteration times.

Appendix F
Geometry of rs-SDR mesh

Property	**Size [m]**
Stator radius	0.067
Rotor radius	0.066
Shaft radius	0.006
Inlet radius	0.01
Outlet radius	0.005
Interdisc spacing (bottom)	0.002
Interdisc spacing (top)	0.002
Rotor thickness	0.008

Table F.1: Geometric properties of the rotor-stator spinning disc reactor as simulated in this study.

In Table F.1, the dimensions of the rs-SDR that was simulated in this study are provided. Using these dimensions, the rotor and shaft of the rs-SDR were designed in SolidWorks. This resulted in the CAD as depicted in Figure F.1. The mesh that was constructed using this CAD is visualised in Figure F.2. In this configuration of the rs-SDR, the shaft is only affecting the bottom gap between rotor and stator. The inlet is located at the bottom stator with a geometry as a concentric circle around the shaft. The outlet is located at the centre of the top stator.

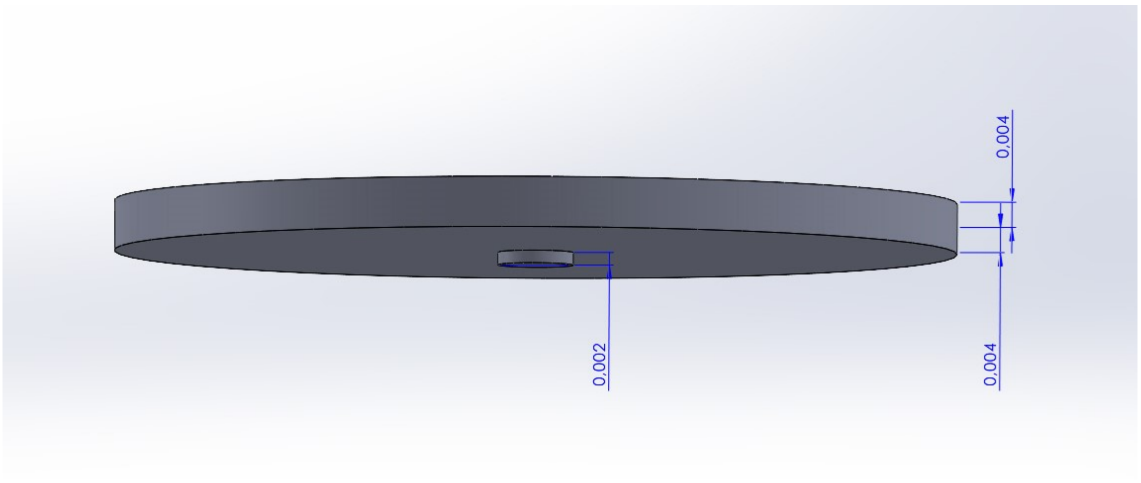


Figure F.1: The CAD of the rotor and shaft of the rotor-stator spinning disc reactor as simulated in this study.

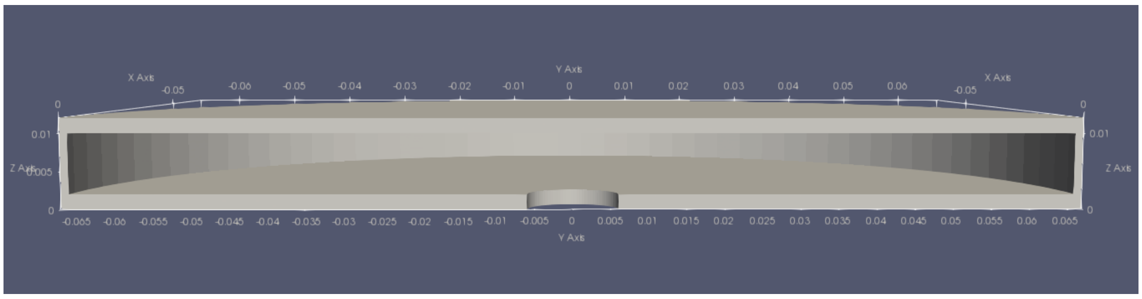


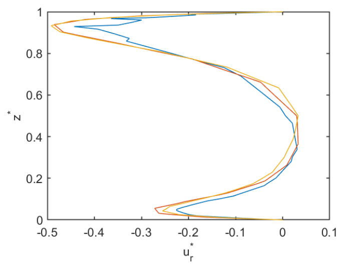
Figure F.2: A bisected view of the mesh that represents the zones of the rotor-stator spinning disc reactor that are open to flow. Note that the inlet and outlet are not visualised in this view, as these are implemented in the model via patches on the mesh.

Appendix G
Grid dependency study rs-SDR

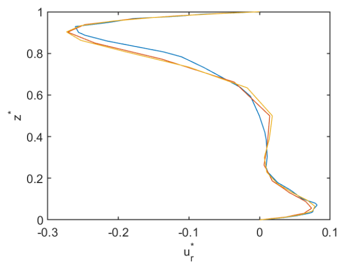
	N_θ	N_r	N_z
Coarse	125	100	90
Middle	200	150	140
Fine	250	200	180

Table G.1: The number of grid cells used in the grid dependency study of the rs-SDR

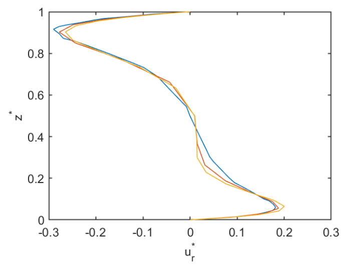
To select an accurate resolution of the grid, a grid dependency study was performed on the rs-SDR simulation as was discussed in Section 3.2. The number of grid cells that was selected for the grid dependency study are depicted in Table G.1. In Figure G.1, the results of the grid dependency study are visualised. The axial velocity profiles between the rotor and the top stator are depicted for three radial positions. From the velocity profiles, it was decided that the coarse mesh was inadequate of capturing the flow behaviour accurately, since spikes and oscillations were observed in the velocity profiles. The relative difference between the middle and the fine mesh was computed for different axial profiles of the radial velocity and the maximum deviation was found to be well below 8 percent. On basis of this and keeping computational costs limited, it was decided that the middle was used in further simulations.



(a) $r = 0.01r_d$



(b) $r = 0.2r_d$



(c) $r = 0.4r_d$

Figure G.1: Grid dependency study on the simulation of the rotor-stator spinning disc reactor simulation at Re $= 2.18 \cdot 10^5$ and $\phi_v = 1 \cdot 10^{-5}$ m^3 s^{-1}. The axial velocity profiles are visualised for the axial gap between rotor and top stator. The blue line denotes the coarse mesh, the red line denotes the middle mesh and the yellow line denotes the fine mesh.

Appendix H Experimental calibrations

Before the residence time distributions in the rs-SDR were experimentally determined, the set-up as described in Section 3.5 was calibrated. For this, the gear pump and UV-VIS spectrometer were calibrated. The pump was calibrated by measuring the mass flow rate as a function of the set gear pump percentage. This calibration curve can be seen in Figure H.1 and an R^2-value bigger than 0.99999 was obtained.

The UV-VIS calibration was performed by circulating a solution with a known concentration of methylene blue through the reactor. At the inlet and the outlet, the UV-VIS spectra were measured after 10 recirculation times of the vessel containing the bulk solution. In order to promote homogenisation of the tracer, the rotational speed of the spinning disc was set to 50 rad/s. As a reference, the spectra were also measured for dark conditions and when circulating water. The absorbance was computed according to Equation H.1, in which N_0 and N_{ref} denote the measured intensity in the dark and for water, respectively. At low concentrations, the absorbance can be related to Beer-Lambert's law, which is expressed in Equation H.2. In Figure H.2, the measured absorbance for different concentrations are displayed. The absorbances were measured for the absorption peaks of methylene blue, which were found to be 292, 613 and 664 nm [103]. It can be seen that Beer-Lambert's law corresponds well to the experimental data at low concentrations, but at higher concentration a deviation was observed from the linear relationship. In Figure H.2, the fitted line was obtained by fitting the experimental data below an absorbance of 1. In this region, an R^2-value above 0.99 was obtained for all measured wavelengths. From this fitting, the molar extinction coefficient was determined according to Equation H.2 with an optical path length of 10 mm. The results of the molar extinction coefficients are displayed in Table H.1. The maximum R^2-value (> 0.999) was found at a wavelength of 613 nm and it can be seen that the linear regime is valid for higher concentrations. This makes this wavelength optimal for the residence time distribution measurements. For these measurements, it was ensured that the maximum concentration stays below $3 \cdot 10^{-4}$ M to ensure operation in the linear regime.

$$A = -\frac{\log(N - N_0)}{\log(N_{ref} - N_0)} \tag{H.1}$$

$$A = \varepsilon_m l_{path} c \tag{H.2}$$

Wavelength [nm]	**Molar extinction coefficient [m^2 mol^{-1}]**	
	Inlet	**outlet**
292	373.7	409.9
613	467.5	521.7
664	756.2	833.7

Table H.1: The molar extinction coefficient of methylene blue as measured by UV-VIS spectroscopy for both the flow cells at the inlet and the outlet.

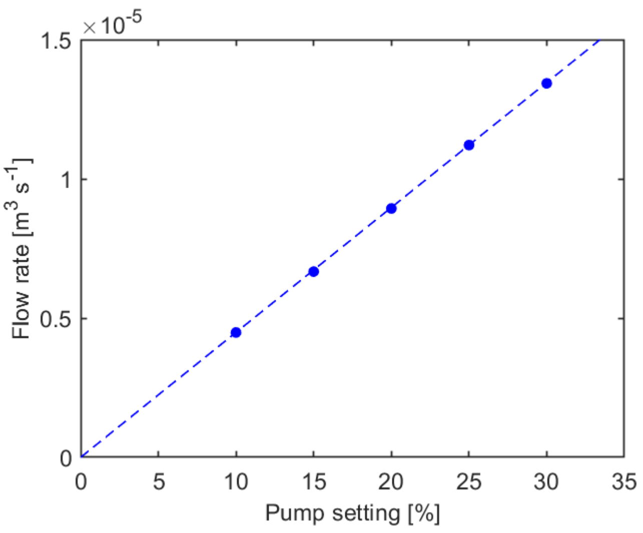


Figure H.1: Calibration curve of the VerderGear V1000 gear pump.

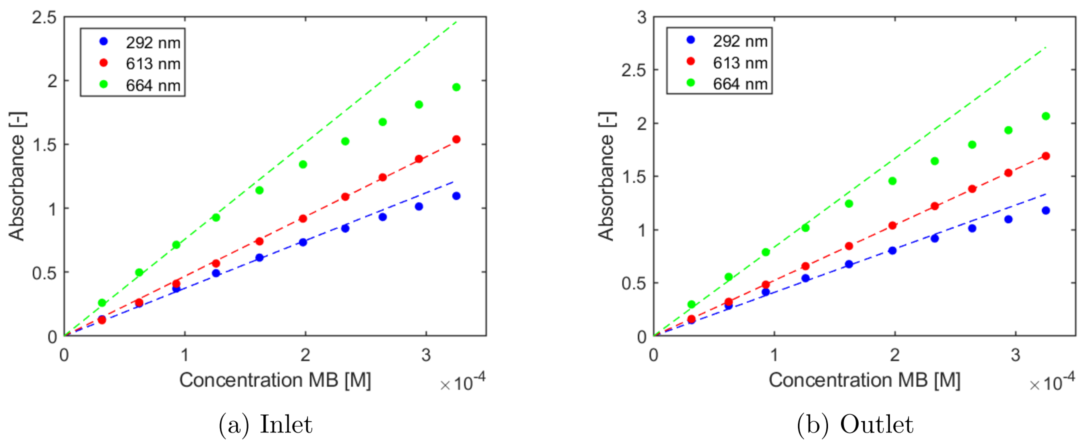


Figure H.2: Calibration curves using the Beer-Lambert law for both the flow cells located at the inlet and the outlet measured at the absorption peaks of methylene blue. The trend lines are included to visualise the linear regimes.

Appendix I
Residence time distribution rs-SDR

In this Appendix, the residence time distributions of the rs-SDR that were studied by means of experiments and simulations are discussed. Firstly, the residence time distribution experiments are discussed. The RTD was measured for 13 different rotational speeds and all experiments were performed in triplo. Using the calibration curve for the concentration of methylene blue, the inlet and outlet concentrations were measured. It was ensured that the injection peak remains below three seconds and the maximum concentration near the injection zone was maintained below $3 \cdot 10^{-4}$ M to stay in the linear regime. In Figure I.1, a typical profile of the normalised inlet and outlet concentrations are depicted. Using the deconvolution strategy as discussed in Section 3.5, accurate fits of the engineering model were obtained with a R^2-value above 0.99 for all experiments. In Figure I.2, the result of a fit is compared to the measured outlet concentration for a typical experiment.

Using MATLAB's gradient function, the gradient of the outlet concentration was computed and the RTD was determined. The resulting RTD was processed using a moving average to filter the fluctuations in the results. Thereafter, the engineering model was fitted to the simulated results. For all turbulent cases, an adequate fit was obtained for all simulations ($R^2 > 0.997$), but the laminar case was found to be unsuccessfully represented by the simulation results. It was found that the engineering model predicts a more steep gradient after the residence time of PFR-zone has been passed compared to the simulation results. This is related to the dispersion that was considered in the simulation. Due to this, also the proposed engineering model with non-ideal dispersion in the throughflow governed regime was fitted to the experimental data and accurate fits were obtained with a R^2-value above 0.99 for all experiments. When the proposed engineering model was fitted to the simulation results, a fit was obtained with R^2-value above 0.996 for all turbulent simulations. In Figure I.3, the simulated RTDs are visualised with the experimental results and the fit of the proposed engineering model. Intermediate steps in the derivation of the non-ideal PFR after a step input are given in Equations I.1-I.4. For more details, [104] can be consulted.

$$\frac{\partial c}{\partial t} + u\frac{\partial c}{\partial z} = D_{ax}\frac{\partial^2 c}{\partial z^2} \tag{I.1}$$

$$\theta = \frac{tu}{L} = \frac{t}{\tau_{PF}} \qquad Z = \frac{z}{L} \qquad Pe = \frac{uL}{D_{ax}} \tag{I.2}$$

$$\frac{\partial c}{\partial \theta} + \frac{\partial c}{\partial Z} = \frac{1}{Pe}\frac{\partial^2 c}{\partial Z^2} \tag{I.3}$$

$$c_{out}(t) = \left(Pe\frac{\tau_{PF}}{4\pi t}\right)^{0.5} \exp\left(-Pe\frac{(1-t/\tau_{PF})^2}{4t/\tau_{PF}}\right) \tag{I.4}$$

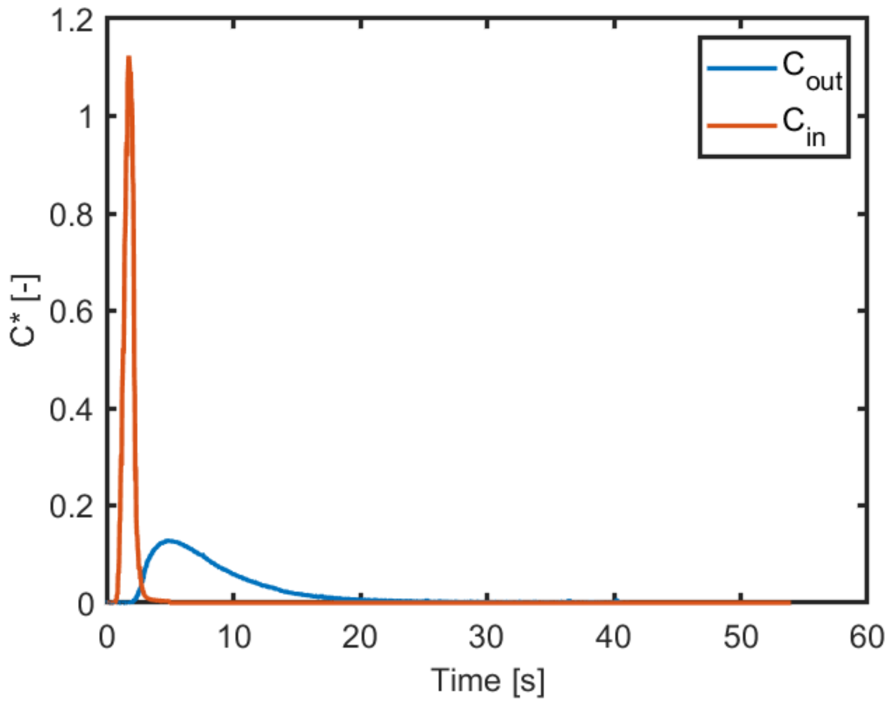


Figure I.1: The normalised inlet and outlet concentrations for a typical experiment. For this case, a rotation speed of 525 RPM was used.

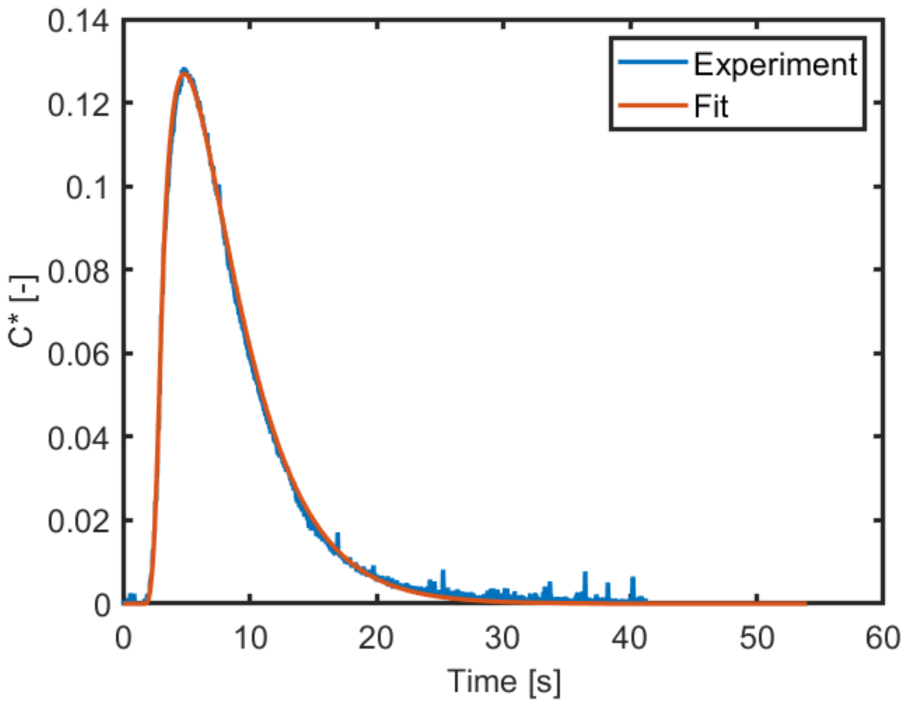


Figure I.2: The measured outlet concentration compared with the result of a typical fitted outlet concentration by means of the deconvolution strategy. For this case, a rotation speed of 525 RPM was used.

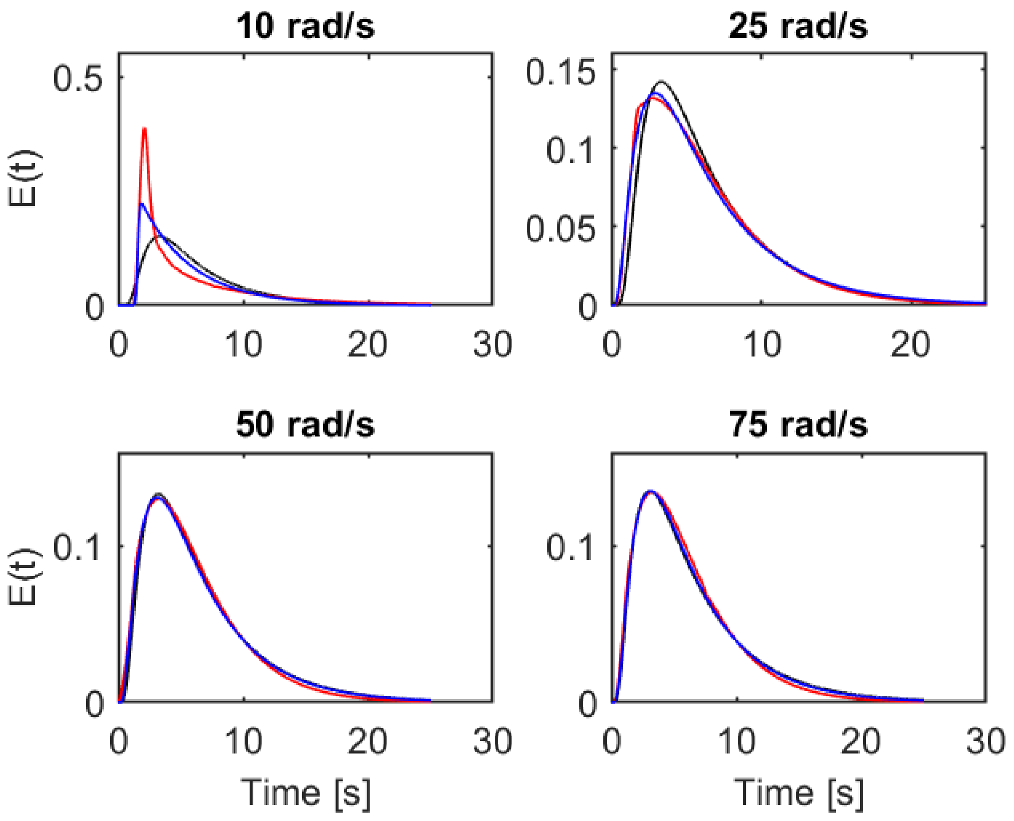


Figure I.3: Comparison of the simulated residence time distribution and the experimentally obtained residence time distribution using the deconvolution with the proposed reactor model. Also, the fit of the proposed reactor model on the experimental RTD is included. The black line denotes the experimental results, the red line denotes the simulation result and the blue line denotes the fit of the proposed reactor model on the simulation results.

Appendix J
Recirculation in the rs-SDR

To study the recirculation at higher radial positions in the rs-SDR, the degree of recirculation was defined as in Equation J.1. In Figure J.1, the degree of recirculation can be seen for the simulations at different rotational speeds. At low radial positions, the degree of recirculation is 1, meaning that there is no recirculation present and the flow regime is throughflow governed. At a certain radial position, the degree of recirculation increases and keeps increasing with increasing radial position. The radial position at which this happens coincides with the definition of the transition radius for the throughflow governed and rotation governed regime. However, the experimentally obtained transition radii are determined by means of the RTD. In order to be a well-mixed zone in the engineering model, the degree of recirculation must be significantly higher than 1. Therefore, the transitional radius as determined from the velocity profiles is underestimated when compared to experimentally obtained values from RTDs.

$$\text{Degree of recirculation} = \frac{\int_0^h 2\pi r|u_r|dh}{\int_0^h 2\pi r u_r dh} \approx \frac{\int_0^h 2\pi r|u_r|dh}{\phi_v} \tag{J.1}$$

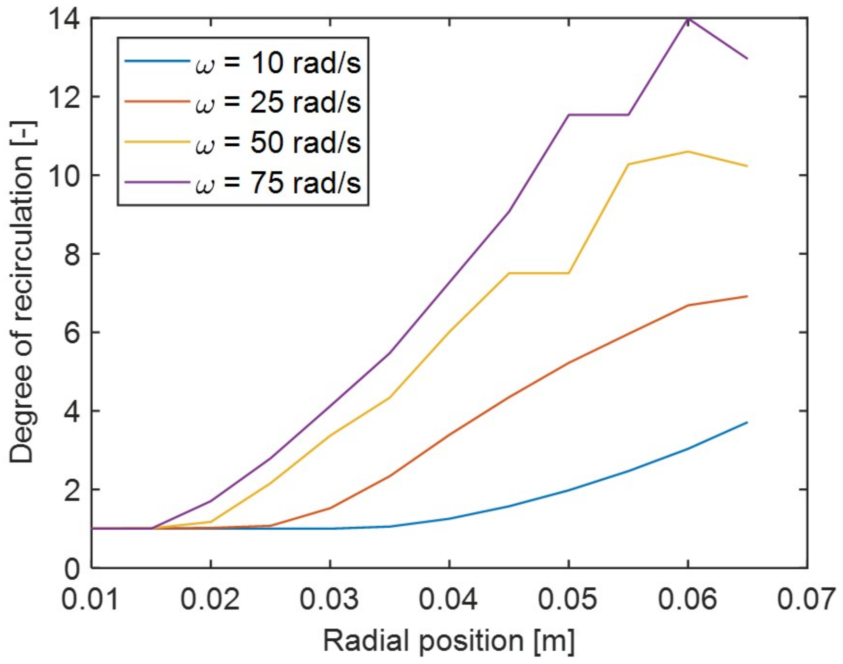


Figure J.1: Comparison of the degree of recirculation as a function of the radial coordinate for different rotational speeds.

Appendix K
Interfacial force calculations

In this Appendix, the methodology of calculating the interfacial forces on the bacterium is presented. The main force was found to be the drag force. Additional forces that are present, such as the virtual mass force, gravitational force and lift force were considered to be insignificant compared to the drag force when determining the magnitude of the forces acting on the bacterium.

$$\boldsymbol{F}_{D,k} = \frac{3}{4}\rho_c \alpha_c \alpha_d C_D \frac{|\boldsymbol{u}_d - \boldsymbol{u}_c| (\boldsymbol{u}_d - \boldsymbol{u}_c)}{d_d} \tag{K.1}$$

$$C_D = \begin{cases} \frac{24\left(1+0.15Re_p^{0.683}\right)}{Re_p} & \text{Re}_p \leq 1000 \\ 0.44 & \text{Re}_p \geq 1000 \end{cases} \tag{K.2}$$

$$Re_p = d_d \frac{|\boldsymbol{u_d} - \boldsymbol{u_c}|}{\nu_c} \tag{K.3}$$

The total drag force was computed as a multiplication of the mean force density as given in Equation K.1 and the bacterial cell volume. By integrating the force density over the area of the bacterial cell and dividing by this cell area, the mean force density was computed.

From the drag force, the resulting torque was computed. This computation is given in Equation K.4. In this computation, the center of rotation was selected to be the center of the bacterium. Similarly to the drag force calculations, the total torque was computed by integration.

$$\mathbf{M} = \mathbf{r} \times \mathbf{F} \tag{K.4}$$

Appendix L
MP-PIC implementation

In this Appendix, some background is provided on the implementation of the MP-PIC model for bacteria-laden flow in the rs-SDR. Firstly, the Liouville equation that is used to compute the particle volume fraction distribution function is elaborated upon, after which the physical models that were used to describe the particle dynamics are discussed.

In Equation L.1, the Liouville equation that describes the particle distribution function is depicted [82]. In this equation, ∇_x and ∇_v represent the divergence operator relative to the position and velocity, respectively. The acceleration $\boldsymbol{a}$ is computed by means of the forces acting on the particles, which are discussed in the remainder of this section.

$$\frac{\partial \theta}{\partial t} + \nabla_x \cdot (\theta \boldsymbol{u}) + \nabla_v \cdot (\theta \boldsymbol{a}) = 0 \tag{L.1}$$

In Equation L.2, the computation of the particle acceleration is shown [82]. It can be seen that the acceleration consists of multiple contributions. The contributions as depicted in Equation L.2 represent (in order of appearance) the contribution due to drag, pressure gradients, gravity, lift force and inter-particle stresses. Additionally, the effect of virtual mass was taken into account.

$$\boldsymbol{a} = C_D(\boldsymbol{u}_c - \boldsymbol{u}_d) + \frac{1}{\rho_p}\nabla p + \mathbf{g} + C_L(\boldsymbol{u}_c - \boldsymbol{u}_d) \times \nabla \times \boldsymbol{u}_c - \frac{1}{\rho_p \alpha_p}\nabla \tau \tag{L.2}$$

Physical models and correlations were used to quantify some of these forces. For the drag force, the non-spherical drag correlation by Haider and Levenspiel [86] was used, which is given in Equations L.3 and L.4. In this correlation, ϕ denotes the sphericity of the bacteria, which was calculated to be 0.655. In Equation L.5, the implementation of the virtual mass is demonstrated. For the virtual mass coefficient C_{vm}, a value of 0.5 was selected. The shear-induced lift force implementation in OpenFOAM using the Saffman-Mei model is provided in Equation L.6 and L.7 [105]. To conclude, the inter-particle stress model by Harris and Crighton is depicted in Equation L.8 [83], in which P_s is the solids pressure and α_{cp} is the close-packed particle volume fraction. For numerical implementation, the singularity is avoided by using a small value for ε_s.

- **Drag: non-spherical drag model**

$$C_D = \frac{24}{Re_p}\Big(1 + \exp\big(2.3288 - 6.4581\phi + 2.4486\phi^2\big) Re_p^{(0.0964+0.5565\phi)}\Big) + \frac{Re_p \exp\big(4.905 - 13.8944\phi + 18.4222\phi^2 - 10.2599\phi^3\big)}{Re_p + \exp(1.4681 + 12.2584\phi - 20.7322\phi^2 + 15.8855\phi^3)} \tag{L.3}$$

$$Re_p = d_d \frac{|\boldsymbol{u_d} - \boldsymbol{u_c}|}{\nu_c} \tag{L.4}$$

- **Virtual mass**

$$F_{vm} = \rho_c V_p C_{vm}\left(\frac{D\boldsymbol{u}_c}{Dt} - \frac{d\boldsymbol{u}_d}{dt}\right) \tag{L.5}$$

- **Lift force: Saffman-Mei**

$$C_L = \begin{cases} 6.46 \cdot \left(1 - 0.3314\sqrt{\frac{Re_w}{2Re_p}}\right)\exp(-0.1Re_p) + 0.3314\sqrt{\frac{Re_w}{2Re_p}} & \mathrm{Re}_p \leq 40 \\ 0.3385 \cdot \sqrt{0.5Re_w} & \mathrm{Re}_p \geq 40 \end{cases} \tag{L.6}$$

$$Re_w = \frac{\rho_c d^2 |\nabla \times \mathbf{u}_c|}{\mu_c} \tag{L.7}$$

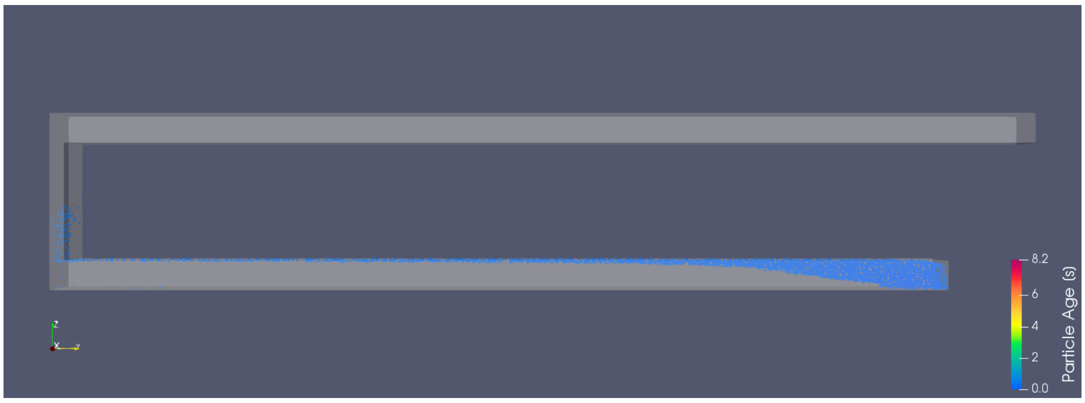
- **Harris-Crighton inter-particle stress model**

$$\tau = P_s \frac{\alpha_p}{\max(\alpha_{cp} - \alpha_p, \varepsilon_s(1 - \alpha_p)} \tag{L.8}$$

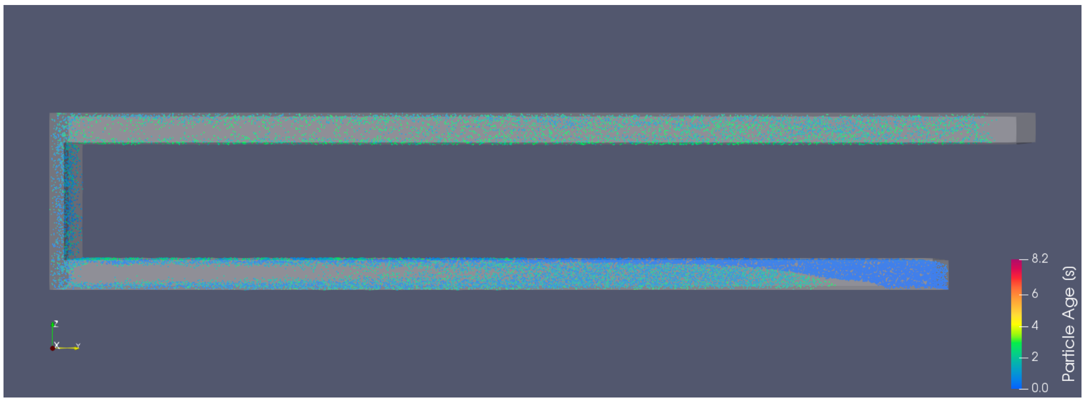
Appendix M Particle dynamics in the rs-SDR

The dynamics of the bacterial cells in the rs-SDR were studied using the MP-PIC model. As discussed in Section 4.4, the dynamics of the bacteria resembles the hydrodynamics in the rs-SDR that were elucidated in this study. In Figure M.1, three overviews are given of the particles in the rs-SDR at different simulation times. For the MP-PIC simulation, a rotational Reynolds number of $2.18 \cdot 10^5$ and a volumetric flow rate of $1 \cdot 10^{-5}$ m^3 s^{-1} was used. The particles are visualised in a bisected slice of the rs-SDR, such that the radial motion of the particles can be studied. The colour coding that is used is the particle age, which is defined as the duration between the current time and the time at which the particle was injected.

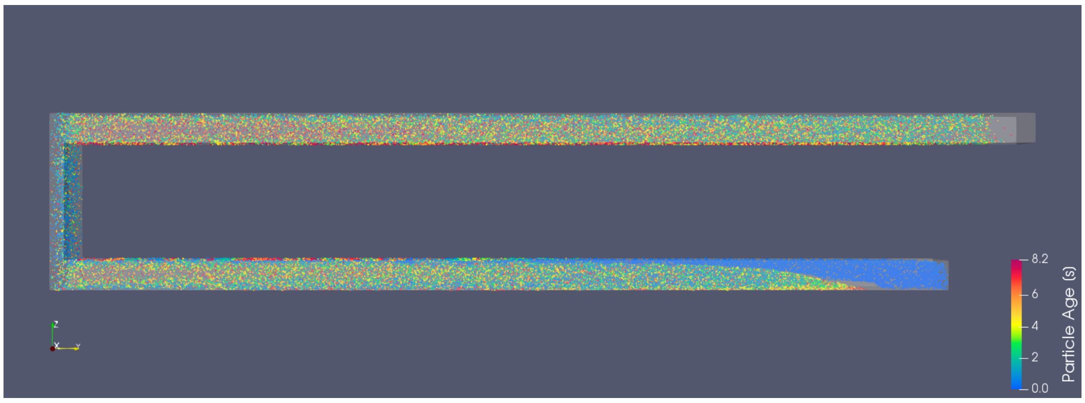
In Figure M.1a, the particles are visualised for a short time. It can be seen that the bacteria move radially outward from the inlet via the Von Kármán boundary layer. In Figure M.1b, it can be seen that recirculation in the boundary layers is occuring at the higher radial positions. In the inviscid core at the rotation governed regime, a lower particle fraction is observed. After more than one residence time, the particles were found to propagate from the turbulent boundary layers to the inviscid core, yielding a homogeneous distribution of particles in the rs-SDR, as visualised in Figure M.1c. From the particle age, a clear transition zone between the rotation-governed regime and the throughflow governed regime can be noticed. At low radial positions near the inlet, the particle age is uniform and low. At higher radial positions, the distribution of particles ages is broad, denoting the mixing in the rotation-governed regime.



(a) $t = 0.05\tau$



(b) $t = 0.5\tau$



(c) $t = 1.4\tau$

Figure M.1: Distribution and particle age of bacteria in the MP-PIC simulation at different simulation times in a bisected slice of the rs-SDR.

Appendix N
Code listings

N.1 rotor-stator cavity

Listing N.1: *Allrun* file of rotor-stator cavity simulation

```
#!/bin/sh
cd ${0%/*} || exit 1           # Run from this directory
. $WM_PROJECT_DIR/bin/tools/RunFunctions     # Source tutorial run functions


####################### The steps to run the simulation ######################

runApplication blockMesh        # Creating the mesh

runApplication topoSet

runApplication createPatch -overwrite

runApplication checkMesh        # Checking the mesh

runApplication decomposePar -force        # Divide the workload over the cores

runApplication mpirun -np 256 pimpleFoam -parallel    # Run the simulation in
    parallel


runApplication reconstructPar -latestTime   # Reconstruct the decomposed folders


#----------------------------------------------------------------------------
```

Listing N.2: *blockMeshDict* file specifying the geometry of the rotor-stator cavity simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1806                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include "../Parameters"
#include "../Calculations"

scale   1;

vertices                 //list of vertices defining the geometry
(
    ($r0_2 $r0_1 0)          //This is point 0. Not 1!
    ($rO_3 $rO_2 0)         //This is point 1
    ($rO_4 $rO_3 0)         //point 2
    ($rO_1 $rO_4 0)         //point 3
    ($rO_2 $rO_1 $h)        //point 4
    ($rO_3 $rO_2 $h)        //point 5
    ($rO_4 $rO_3 $h)        //point 6
    ($rO_1 $rO_4 $h)        //point 7
    ($rI_2 $rI_1 0)         //point 8
    ($rI_3 $rI_2 0)         //Point 9
    ($rI_4 $rI_3 0)         //point 10
    ($rI_1 $rI_4 0)         //point 11
    ($rI_2 $rI_1 $h)        //point 12
    ($rI_3 $rI_2 $h)        //point 13
    ($rI_4 $rI_3 $h)        //point 14
    ($rI_1 $rI_4 $h)        //point 15
);

blocks                  //defining the blocks which the geometry is made out of
(

    hex (0 1 9 8 4 5 13 12) ($Ntheta_ $Nr $Nz) simpleGrading (1 ((50 50 5) (50 50
    0.2)) ((50 50 10) (50 50 0.1)))
    hex (1 2 10 9 5 6 14 13) ($Ntheta_ $Nr $Nz) simpleGrading (1 ((50 50 5) (50 50
    0.2)) ((50 50 10) (50 50 0.1)))
    hex (2 3 11 10 6 7 15 14) ($Ntheta_ $Nr $Nz) simpleGrading (1 ((50 50 5) (50 50
     0.2)) ((50 50 10) (50 50 0.1)))
    hex (3 0 8 11 7 4 12 15) ($Ntheta_ $Nr $Nz) simpleGrading (1 ((50 50 5) (50 50
    0.2)) ((50 50 10) (50 50 0.1)))

);


edges
(
    arc 0 1 (0 $R3_ 0)
    arc 4 5 (0 $R3_ $h)
    arc 1 2 ($R3 0 0)
    arc 5 6 ($R3 0 $h)
    arc 2 3 (0 $R3 0)
    arc 6 7 (0 $R3 $h)
    arc 3 0 ($R3_ 0 0)
    arc 7 4 ($R3_ 0 $h)

    arc 9 8 (0 $R1_ 0)
    arc 13 12 (0 $R1_ $h)
    arc 10 9 ($R1 0 0)
    arc 14 13 ($R1 0 $h)
    arc 11 10 (0 $R1 0)
    arc 15 14 (0 $R1 $h)
    arc 8 11 ($R1_ 0 0)
    arc 12 15 ($R1_ 0 $h)
);


boundary
(
    // Define the boundaries of the 4 block system (easiest way to understand is by
     drawing the system)
    rotatingWalls
    {
      type wall;
      faces
      (
        (0 8 9 1)
        (1 9 10 2)
        (2 10 11 3)
        (3 11 8 0)
      );
    }

    fixedWalls
    {
      type wall;
      faces
      (
        (4 12 13 5)
        (5 13 14 6)
        (6 14 15 7)
        (7 15 12 4)
      );
    }

    back
    {
      type wall;
      faces
      (
        (8 12 13 9)
        (9 13 14 10)
        (10 14 15 11)
        (11 15 12 8)
      );
    }

    front
    {
      type wall;
      faces
      (
        (0 4 5 1)
        (1 5 6 2)
        (2 6 7 3)
        (3 7 4 0)
      );
    }

 );

// ************************************************************************* //
```

Listing N.3: *fvSolution* file specifying the solvers of the rotor-stator cavity simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1806                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p
    {
        solver           GAMG;
        smoother       GaussSeidel
        preconditioner    FDIC;
        tolerance        1e-06;
        relTol           0.05;
        cacheAgglomeration  yes;
        nPreSweeps       2;
        nPostSweeps      2;
    }

    pFinal
    {
        $p;
        relTol          0;
    }

    "(U|nut|nuTilda|flm|fmm)"
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        tolerance       1e-05;
        relTol          0;
    }

    UFinal
    {
        $U;
        relTol          0;
    }

}

PIMPLE
{
    nCorrectors          1;
    nNonOrthogonalCorrectors  0;
    pRefCell           0;
    pRefValue          0;
    nOuterCorrectors      5;
    turbOnFinalIterOnly     no;

residualControl
    {
      U   1e-05;
      p   5e-04;
    }
}





// ************************************************************************* //
```

Listing N.4: *fvSchemes* file specifying the schemes of the rotor-stator cavity simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1806                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         CrankNicolson 0.9;
}

gradSchemes
{
    default    Gauss cubic;
    grad(p)    Gauss cubic;
    grad(U)    Gauss cubic;
    grad(nut) Gauss cubic;
}

```

```
divSchemes
{
    default              Gauss vanLeer;
    div(phi,U)             Gauss vanLeer;
    div(phi,nut)           Gauss vanLeer;
    div(yPhi,yWall)          Gauss vanLeer;
    div((nuEff*dev2(T(grad(U)))))   Gauss cubic;
}

laplacianSchemes
{
    default       Gauss cubic corrected;
    laplacian(yWall)  Gauss cubic corrected;
}

interpolationSchemes
{
    default         cubic;
}

snGradSchemes
{
    default         cubic corrected;
}


// ************************************************************************* //
```

Listing N.5: Initial and boundary conditions of the velocity for the rotor-stator cavity simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1906                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include "../Parameters"  // Include the parameters
#include "../Calculations"  // Include the Calculation for the parameters

dimensions      [0 1 -1 0 0 0 0];

internalField #codeStream
  {
    codeInclude
    #{
    #include "fvCFD.H"
    #};

    codeOptions
    #{
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude
    #};

    codeLibs
    #{
    -lmeshTools \
    -lfiniteVolume
    #};

    code
    #{
    const IOdictionary& d = static_cast<const IOdictionary&>(dict);
    const fvMesh& mesh = refCast<const fvMesh>(d.db());

    vectorField v(mesh.nCells());

    scalar uB = 0.4;  // Relative bulk velocity

    forAll(v, i)
    {
    const scalar x = mesh.C()[i][0];
    const scalar y = mesh.C()[i][1];

    scalar r = sqrt(pow(x,2)+pow(y,2));

    if (r <= $R2 ){

                scalar vT = r * uB * $w;

                scalar vY = vT * r / (x + pow(y,2) / x);

                scalar vX = - y / x * vY;

        // Get some random perturbation for the velocity to initialize

                scalar vX_rand = ( rand() * 1e-9 ) - 1;
                scalar vY_rand = ( rand() * 1e-9 ) - 1;
                scalar vZ_rand = ( rand() * 1e-10 ) - 0.11;

                scalar vX_perturb = vX + vX_rand;
                scalar vY_perturb = vY + vY_rand;
                scalar vZ_perturb = vZ_rand;

                v[i] = vector(vX_perturb,vY_perturb,vZ_perturb);

    }
                else
    {
    //Velocity 0 at R2 and wRi at R3 linearly
                scalar vT = uB *(-(r - $R2) / ($R3 - $R2) * $w *$R2 + $w * $R2);

                scalar vY = vT * r / (x + pow(y,2) / x);

                scalar vX = - y / x * vY;

                scalar vX_rand = ( rand() * 1e-9 ) - 1;
                scalar vY_rand = ( rand() * 1e-9 ) - 1;
                scalar vZ_rand = ( rand() * 1e-10 ) - 0.11;

                scalar vX_perturb = vX + vX_rand;
                scalar vY_perturb = vY + vY_rand;
                scalar vZ_perturb = vZ_rand;

                v[i] = vector(vX_perturb,vY_perturb,vZ_perturb);

                }

    }

    writeEntry(os, "", v);

    #};
  };


boundaryField
{
    rotor
    {
        type            rotatingWallVelocity;
        origin          (0 0 0);
        axis            (0 0 1);
        omega           constant $w;
    }

    stator
    {
        type            noSlip;
    }

    front
    {
        type            noSlip;
    }

    back
    {
        type            rotatingWallVelocity;
        origin          (0 0 0);
        axis            (0 0 1);
        omega           constant $w;
    }

    inlet
    {
        type             fixedValue;
        value    #codeStream
        {
        codeInclude
        #{
    #include "fvCFD.H"
  #};

  codeOptions
        #{
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude
  #};

  codeLibs
  #{
    -lmeshTools \
    -lfiniteVolume
  #};

        code
        #{
    const IOdictionary& d = static_cast<const IOdictionary&>
    (
      dict.parent().parent()
    );

    const fvMesh& mesh = refCast<const fvMesh>(d.db());
    const label id = mesh.boundary().findPatchID("inlet");
    const fvPatch& patch = mesh.boundary()[id];

    vectorField U(patch.size(), vector(0, 0, 0));

    forAll(U,i)
    {
      const scalar x = patch.Cf()[i][0];
      const scalar y = patch.Cf()[i][1];

      scalar r = sqrt(pow(x,2)+pow(y,2));

      //Velocity linearly between zero at R4 and wRi at R1
      scalar vT = -(r - $R1) / ($R4 - $R1) * $w *$R1 + $w * $R1;

      scalar vY = vT * r / (x + pow(y,2) / x);

      scalar vX = - y / x * vY;

      U[i] = vector(vX,vY,0);
    }

    writeEntry(os,"",U);
       #};

      };
    }

    outlet
    {
        type            fixedValue;
        value           #codeStream
        {
        codeInclude
        #{
                #include "fvCFD.H"
        #};

        codeOptions
        #{
                -I$(LIB_SRC)/finiteVolume/lnInclude \
                -I$(LIB_SRC)/meshTools/lnInclude
        #};

        codeLibs
        #{
                -lmeshTools \
                -lfiniteVolume
        #};

        code
        #{
                const IOdictionary& d = static_cast<const IOdictionary&>
                (
                        dict.parent().parent()
                );

                const fvMesh& mesh = refCast<const fvMesh>(d.db());
                const label id = mesh.boundary().findPatchID("outlet");
                const fvPatch& patch = mesh.boundary()[id];

                vectorField U(patch.size(), vector(0, 0, 0));


    forAll(U,i)
    {

    const scalar x =  patch.Cf()[i][0];
    const scalar y =  patch.Cf()[i][1];

    scalar r = sqrt(pow(x,2)+pow(y,2));

    //Velocity linearly from 0 at R2 to wRi at R3

    scalar vT = -(r - $R2) / ($R3 - $R2) * $w *$R2 + $w * $R2;

    scalar vY = vT * r / (x + pow(y,2) / x);

    scalar vX = - y / x * vY;

    U[i] = vector(vX,vY,0);
    }
    writeEntry(os,"",U);
       #};
    };
}


// ************************************************************************* //
```

Listing N.6: Initial and boundary conditions of the pressure for the rotor-stator cavity simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1906                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0];


internalField   uniform 0;

boundaryField
{
    rotor
    {
        type    zeroGradient;
    }

    stator
    {
        type    zeroGradient;
    }

    front
    {
        type    zeroGradient;
    }

    back
    {
        type    zeroGradient;
    }

    inlet
    {
        type    zeroGradient;
    }

    outlet
    {
        type    zeroGradient;
    }
}


// ************************************************************************* //
```

Listing N.7: Initial and boundary conditions of the turbulent viscosity for the rotor-stator cavity simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1906                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      nut;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include "../Parameters"        // Include the parameters

dimensions      [0 2 -1 0 0 0 0];
```

```

internalField   uniform $nu;

boundaryField
{
    rotor
    {
        type            fixedValue;
        value           uniform 0;
    }

    stator
    {
        type            fixedValue;
        value           uniform 0;
    }

    front
    {
        type            fixedValue;
        value           uniform 0;
    }

    back
    {
        type            fixedValue;
        value           uniform 0;
    }

    inlet
    {
        type             calculated;
        value            uniform 0;
    }

    outlet
    {
        type             calculated;
        value            uniform 0;
    }
}


// ************************************************************************* //
```

N.2 rs-SDR simulation

Listing N.8: *Allrun* file of rs-SDR simulation

```
#!/bin/sh
cd ${0%/*} || exit 1           # Run from this directory
. $WM_PROJECT_DIR/bin/tools/RunFunctions     # Source tutorial run functions

################### The steps to run the simulation ###################

runApplication blockMesh        # Creating the mesh

runApplication surfaceFeatures                    # Extract edges from trisurface

runApplication snappyHexMesh -overwrite

runApplication topoSet                                  # Define additional faces

runApplication createPatch -overwrite                   # Make additional patches

runApplication checkMesh  #Check the mesh

runApplication decomposePar -force    # Divide the workload over the cores

runApplication mpirun -np 256 pimpleFoam -parallel    # Run the simulation in
    parallel with 256 cores with pisoFoam
```

```

runApplication reconstructPar -latestTime   # Reconstruct the decomposed folders

#--------------------------------------------------------------------
```

Listing N.9: *blockMeshDict* file specifying the geometry of the rs-SDR simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include "../Parameters"
#include "../Calculations"

convertToMeters 1;  //This gives you the scale: 0.001 would mean all units in this
    file are [mm]

Rd_ #calc "$Rd * -1.0";

vertices    //list of vertices defining the geometry
(
    ($R2 $R1 0)   //This is point 0. Not 1!
    ($R3 $R2 0)   //This is point 1
    ($R4 $R3 0)   //point 2
    ($R1 $R4 0)   //point 3
    ($R2 $R1 $h)  //point 4
    ($R3 $R2 $h)  //point 5
    ($R4 $R3 $h)  //point 6
    ($R1 $R4 $h)  //point 7
    (0 0 0)   //point 8
    (0 0 $h)    //point 9
);

blocks      //defining the block which the geometry is made out of
(

    hex (0 1 8 8 4 5 9 9) ($Ntheta_ $Nr $Nz) simpleGrading (1 ((1 10 10)(60 60 1)(6
     3 1)) ((1 5 20)(1 5 0.1)(8 60 1)(1 5 10)(1 5 0.05)))
    hex (1 2 8 8 5 6 9 9) ($Ntheta_ $Nr $Nz) simpleGrading (1 ((1 10 10)(60 60 1)(6
     3 1)) ((1 5 20)(1 5 0.1)(8 60 1)(1 5 10)(1 5 0.05)))
    hex (2 3 8 8 6 7 9 9) ($Ntheta_ $Nr $Nz) simpleGrading (1 ((1 10 10)(60 60 1)(6
     3 1)) ((1 5 20)(1 5 0.1)(8 60 1)(1 5 10)(1 5 0.05)))
    hex (3 0 8 8 7 4 9 9) ($Ntheta_ $Nr $Nz) simpleGrading (1 ((1 10 10)(60 60 1)(6
     3 1)) ((1 5 20)(1 5 0.1)(8 60 1)(1 5 10)(1 5 0.05)))

);

edges
(
    arc 0 1 (0 $Rd_ 0)
    arc 4 5 (0 $Rd_ $h)
    arc 1 2 ($Rd 0 0)
    arc 5 6 ($Rd 0 $h)
    arc 2 3 (0 $Rd 0)
    arc 6 7 (0 $Rd $h)
    arc 3 0 ($Rd_ 0 0)
    arc 7 4 ($Rd_ 0 $h)
);

boundary
(
    walls //choose a name for the boundary
    {
        type wall;  //define the type of the boundary
        faces
        (
          (0 4 5 1)
          (2 6 7 3)
          (2 1 5 6)
          (3 7 4 0)
        );
    }


    top
    {
        type patch;
        faces
        (
          (5 9 9 4)
          (6 9 9 5)
          (7 9 9 6)
          (4 9 9 7)
      );
    }


    bottom
    {
        type patch;
        faces
        (
          (0 8 8 1)
          (1 8 8 2)
          (2 8 8 3)
          (3 8 8 0)
        );
    }
);


// ************************************************************************* //
```

Listing N.10: *fvSolution* file specifying the solvers of the rs-SDR simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p
    {
        solver          GAMG;
        tolerance       1e-06;
        relTol          0.05;
        smoother        GaussSeidel;
        preconditioner  FDIC;
        cacheAgglomeration   yes;
        nPreSweeps      2;
        nPostSweeps     2;
    }

    pFinal
    {
        $p;
        relTol          0;
    }

    "(U|nut|nuTilda|Phi)"
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        tolerance       1e-05;
        relTol          0.1;
    }

    UFinal
    {
        $U;
        relTol  0;
    }
}

PIMPLE
{
    nCorrectors     1;
    nNonOrthogonalCorrector 0;
    pRefCell      0;
    pRefValue     0;
    nOuterCorrectors    5;
    turbOnFinalIterOnly   no;

    residualControl
    {
        U 1e-05;
        p 5e-04;
   }
}

cache
{
  grad(U);
}



// ************************************************************************* //
```

Listing N.11: *fvSchemes* file specifying the schemes of the rs-SDR simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1806                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         CrankNicolson 0.9;
}

gradSchemes
{
    default     Gauss cubic;
    grad(p)     Gauss cubic;
    grad(U)     Gauss cubic;
    grad(Phi)   Gauss cubic;
    grad(nut)   Gauss cubic;
}

divSchemes
{
    default             Gauss cubic;
    div(phi,U)            Gauss cubic;
    div(div(phi,U))           Gauss cubic;
    div(phi,nut)          Gauss cubic;
    div(yPhi,yWall)         Gauss cubic;
    div((nuEff*dev2(T(grad(U))))) Gauss cubic;
}

laplacianSchemes
{
    default       Gauss cubic corrected;
    laplacian(1,Phi)    Gauss cubic corrected;
    laplacian(yWall)  Gauss cubic corrected;
}

interpolationSchemes
{
    default         cubic;
}

snGradSchemes
{
    default         cubic corrected;
}


// ************************************************************************* //
```

Listing N.12: *snappyHexMeshDict* file specifying the construction of the mesh of the rs-SDR simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      autoHexMeshDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include "../Parameters"        // Include the parameters
#include "../Calculations"      // Include the calculations for the constants

z1   #calc "$h - 0.0005";
r1   #calc "$Ri + 0.002";

// Which of the steps to run
castellatedMesh true;
snap            false;
addLayers       false;


// Geometry. Definition of all surfaces. All surfaces are of class
// searchableSurface.
// Surfaces are used
// - to specify refinement for any mesh cell intersecting it
// - to specify refinement for any mesh cell inside/outside/near
// - to 'snap' the mesh boundary to the surface
geometry
{
    discShaft_2mm.stl
    {
        type triSurfaceMesh;
        name discShaft_2mm;
    }

    // Regions for additional refinement
    outletRefinement
    {
        type searchableCylinder;
        point1 (0 0 $h);
        point2 (0 0 $z1);
        radius $Ro;
    }

};



// Settings for the castellatedMesh generation.
castellatedMeshControls
{

    // Refinement parameters
    // ~~~~~~~~~~~~~~~~~~~~~

    // If local number of cells is >= maxLocalCells on any processor
    // switches from from refinement followed by balancing
    // (current method) to (weighted) balancing before refinement.
    maxLocalCells 10000000;

    // Overall cell limit (approximately). Refinement will stop immediately
    // upon reaching this number so a refinement level might not complete.
    // Note that this is the number of cells before removing the part which
    // is not 'visible' from the keepPoint. The final number of cells might
    // actually be a lot less.
    maxGlobalCells 100000000;

    // The surface refinement loop might spend lots of iterations refining just a
    // few cells. This setting will cause refinement to stop if <= minimumRefine
    // are selected for refinement. Note: it will at least do one iteration
    // (unless the number of cells to refine is 0)
    minRefinementCells 2;

    // Number of buffer layers between different levels.
    // 1 means normal 2:1 refinement restriction, larger means slower
    // refinement.
    nCellsBetweenLevels 1;



    // Explicit feature edge refinement
    // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    // Specifies a level for any cell intersected by its edges.
    // This is a featureEdgeMesh, read from constant/triSurface for now.
    features
    (
        {
            file "discShaft_2mm.eMesh";
            level 0;
        }
    );



    // Surface based refinement
    // ~~~~~~~~~~~~~~~~~~~~~~~~

    // Specifies two levels for every surface. The first is the minimum level,
```

```
    // every cell intersecting a surface gets refined up to the minimum level.
    // The second level is the maximum level. Cells that 'see' multiple
    // intersections where the intersections make an
    // angle > resolveFeatureAngle get refined up to the maximum level.

    refinementSurfaces
    {
        discShaft_2mm
        {
            // Surface-wise min and max refinement level
            level (0 0);
        }
    }

    resolveFeatureAngle 20;


    // Region-wise refinement
    // ~~~~~~~~~~~~~~~~~~~~~~

    // Specifies refinement level for cells in relation to a surface. One of
    // three modes
    // - distance. 'levels' specifies per distance to the surface the
    //   wanted refinement level. The distances need to be specified in
    //   descending order.
    // - inside. 'levels' is only one entry and only the level is used. All
    //   cells inside the surface get refined up to the level. The surface
    //   needs to be closed for this to be possible.
    // - outside. Same but cells outside.

    refinementRegions
    {
        outletRefinement
        {
      mode inside;
      levels ((0 0));
        }
    }


    // Mesh selection
    // ~~~~~~~~~~~~~~

    // After refinement patches get added for all refinementSurfaces and
    // all cells intersecting the surfaces get put into these patches. The
    // section reachable from the locationInMesh is kept.
    // NOTE: This point should never be on a face, always inside a cell, even
    // after refinement.
    // This is an outside point locationInMesh (-0.033 -0.033 0.0033);
    locationInMesh (0 0.0665 0.005); // Inside point

    // Whether any faceZones (as specified in the refinementSurfaces)
    // are only on the boundary of corresponding cellZones or also allow
    // free-standing zone faces. Not used if there are no faceZones.
    allowFreeStandingZoneFaces true;
}



// Settings for the snapping.
snapControls
{
    //- Number of patch smoothing iterations before finding correspondence
    //  to surface
    nSmoothPatch 4;

    //- Relative distance for points to be attracted by surface feature point
    //  or edge. True distance is this factor times local
    //  maximum edge length.
    tolerance 4;

    //- Number of mesh displacement relaxation iterations.
    nSolveIter 5;
```

```

    //- Maximum number of snapping relaxation iterations. Should stop
    //  before upon reaching a correct mesh.
    nRelaxIter 5;

    // Feature snapping

        //- Number of feature edge snapping iterations.
        //  Leave out altogether to disable.
        nFeatureSnapIter 5;

        //- Detect (geometric) features by sampling the surface
        implicitFeatureSnap true;

        //- Use castellatedMeshControls::features
        explicitFeatureSnap false;

        //- Detect features between multiple surfaces
        //  (only for explicitFeatureSnap, default = false)
        multiRegionFeatureSnap false;
}



// Settings for the layer addition.
addLayersControls
{
    // Are the thickness parameters below relative to the undistorted
    // size of the refined cell outside layer (true) or absolute sizes (false).
    relativeSizes true;

    // Per final patch (so not geometry!) the layer information
    layers
    {
        "discShaft_2mm"
        {
      nSurfaceLayers 0;
        }
    }

    // Expansion factor for layer mesh
    expansionRatio 1.5;


    // Wanted thickness of final added cell layer. If multiple layers
    // is the thickness of the layer furthest away from the wall.
    // See relativeSizes parameter.
    finalLayerThickness 0.7;

    // Minimum thickness of cell layer. If for any reason layer
    // cannot be above minThickness do not add layer.
    // See relativeSizes parameter.
    minThickness 0.05;

    // If points get not extruded do nGrow layers of connected faces that are
    // also not grown. This helps convergence of the layer addition process
    // close to features.
    nGrow 0;


    // Advanced settings

    // When not to extrude surface. 0 is flat surface, 90 is when two faces
    // are perpendicular
    featureAngle 30;

    // Maximum number of snapping relaxation iterations. Should stop
    // before upon reaching a correct mesh.
    nRelaxIter 5;

    // Number of smoothing iterations of surface normals
    nSmoothSurfaceNormals 1;

    // Number of smoothing iterations of interior mesh movement direction
    nSmoothNormals 3;

    // Smooth layer thickness over surface patches
    nSmoothThickness 2;

    // Stop layer growth on highly warped cells
    maxFaceThicknessRatio 0.5;

    // Reduce layer growth where ratio thickness to medial
    // distance is large
    maxThicknessToMedialRatio 0.3;

    // Angle used to pick up medial axis points
    minMedianAxisAngle 90;

    // Create buffer region for new layer terminations
    nBufferCellsNoExtrude 0;


    // Overall max number of layer addition iterations. The mesher will exit
    // if it reaches this number of iterations; possibly with an illegal
    // mesh.
    nLayerIter 30;

    // Max number of iterations after which relaxed meshQuality controls
    // get used. Up to nRelaxIter it uses the settings in meshQualityControls,
    // after nRelaxIter it uses the values in meshQualityControls::relaxed.
    nRelaxedIter 10;
}



// Generic mesh quality settings. At any undoable phase these determine
// where to undo.
meshQualityControls
{
    //- Maximum non-orthogonality allowed. Set to 180 to disable.
    maxNonOrtho 65;

    //- Max skewness allowed. Set to <0 to disable.
    maxBoundarySkewness 20;
    maxInternalSkewness 4;

    //- Max concaveness allowed. Is angle (in degrees) below which concavity
    //  is allowed. 0 is straight face, <0 would be convex face.
    //  Set to 180 to disable.
    maxConcave 80;

    //- Minimum pyramid volume. Is absolute volume of cell pyramid.
    //  Set to a sensible fraction of the smallest cell volume expected.
    //  Set to very negative number (e.g. -1E30) to disable.
    minVol 1e-13;

    //- Minimum quality of the tet formed by the face-centre
    //  and variable base point minimum decomposition triangles and
    //  the cell centre.  Set to very negative number (e.g. -1E30) to
    //  disable.
    //     <0 = inside out tet,
    //      0 = flat tet
    //      1 = regular tet
    minTetQuality -1e30;

    //- Minimum face area. Set to <0 to disable.
    minArea -1;

    //- Minimum face twist. Set to <-1 to disable. dot product of face normal
    //- and face centre triangles normal
    minTwist 0.05;

    //- minimum normalised cell determinant
    //- 1 = hex, <= 0 = folded or flattened illegal cell
    minDeterminant 0.001;

    //- minFaceWeight (0 -> 0.5)
    minFaceWeight 0.05;

    //- minVolRatio (0 -> 1)
    minVolRatio 0.01;

    //must be >0 for Fluent compatibility
    minTriangleTwist -1;

    //- if >0 : preserve single cells with all points on the surface if the
    //  resulting volume after snapping (by approximation) is larger than
    //  minVolCollapseRatio times old volume (i.e. not collapsed to flat cell).
    //  If <0 : delete always.
    //minVolCollapseRatio 0.5;


    // Advanced

    //- Number of error distribution iterations
    nSmoothScale 4;
    //- amount to scale back displacement at error points
    errorReduction 0.75;



    // Optional : some meshing phases allow usage of relaxed rules.
    // See e.g. addLayersControls::nRelaxedIter.
    relaxed
    {
        //- Maximum non-orthogonality allowed. Set to 180 to disable.
        maxNonOrtho 75;
    }
}


// Advanced

// Flags for optional output
// 0 : only write final meshes
// 1 : write intermediate meshes
// 2 : write volScalarField with cellLevel for postprocessing
// 4 : write current intersections as .obj files
debug 0;


// Merge tolerance. Is fraction of overall bounding box of initial mesh.
// Note: the write tolerance needs to be higher than this.
mergeTolerance 1E-6;


// ************************************************************************* //
```

Listing N.13: Initial and boundary conditions of the velocity for the rs-SDR simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include "$FOAM_CASE/Parameters"        // Include the parameters
#include "$FOAM_CASE/Calculations"      // Include the Calculation for the
    parameters
```

```

dimensions      [0 1 -1 0 0 0 0];

internalField #codeStream
{
  codeInclude
  #{
    #include "fvCFD.H"
  #};

  codeOptions
  #{
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude
  #};

  codeLibs
  #{
    -lmeshTools \
    -lfiniteVolume
  #};

  code
  #{
    const IOdictionary& d = static_cast<const IOdictionary&>(dict);
    const fvMesh& mesh = refCast<const fvMesh>(d.db());
    vectorField v(mesh.nCells());

  forAll(v, i)
  {

      const scalar x1 = mesh.C()[i][0];
      const scalar y1 = mesh.C()[i][1];
      const scalar z = mesh.C()[i][2];
      const scalar r = sqrt(pow(x1,2)+pow(y1,2));
      float x;
      float y;

    if (x1 == 0 && y1 == 0){

            x = 0.000001;
            y = 0.000001;

    }
    else if (x1 != 0 && y1 == 0){

            x = x1;
            y = 0.000001;

    }
    else if (x1 == 0 && y1 != 0){

            x = 0.000001;
            y = y1;

    }
    else{

            x = x1;
            y = y1;

    }


    if (r <= $Rr ){

      if (z <= $h2 ){
        scalar vT = r * $uB * $w;
        scalar vR = $phi / (2 * $pi * r * $h1);

        scalar vY = (vR * (y / r )) + (vT * (x / r ));
        scalar vX = (vR * (x / r )) - (vT * (y / r ));

        v[i] = vector(vX,vY,0);
       }


    else{
        scalar vT = r * $uB * $w;
        scalar vR = - $phi / (2 * $pi * r * $h2);

        scalar vY = (vR * (y / r )) + (vT * (x / r ));
        scalar vX = (vR * (x / r )) - (vT * (y / r ));
        v[i] = vector(vX,vY,0);
        }

          }
    else{

          //Velocity 0 at R2 and wRi at R3 linearly
          scalar vT = $uB * (-(r - $Rr) / ($Rd - $Rr) * $w * $Rr + $w * $Rr);
          scalar vY = vT * r / (x + pow(y,2) / x);
          scalar vX = - y / x * vY;
          //Initialize upward velocity
          scalar vZ = $phi / (($pi * $Rd * $Rd) - ($pi * $Rr * $Rr));
          v[i] = vector(vX,vY,vZ);
        }

  }

  writeEntry(os, "", v);

  #};
};

boundaryField
{
    walls
    {
        type    noSlip;
    }

    top
    {
        type    noSlip;
    }

    bottom
    {
        type    noSlip;
    }

    topWall
    {
        type    noSlip;
    }

    bottomWall
    {
        type    noSlip;
    }

    discShaft_2mm
    {
        type            rotatingWallVelocity;
        origin          (0 0 0);
        axis            (0 0 1);
        omega           constant $w;
    }

    inlet
    {
        type    fixedValue;
        value   uniform (0 0 $v_in);
    }

    outlet
    {
        type         inletOutlet;
        value        uniform (0 0 0);
        inletValue   uniform (0 0 0);
    }

}

// ************************************************************************* //
```

Listing N.14: Initial and boundary conditions of the pressure for the rs-SDR simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include "$FOAM_CASE/Parameters"        // Include the parameters
#include "$FOAM_CASE/Calculations"      // Include the Calculation for the
    parameters

dimensions      [0 2 -2 0 0 0 0];

internalField uniform 0;

boundaryField
{
    walls
    {
        type      zeroGradient;
    }

    top
    {
        type      zeroGradient;
    }

    bottom
    {
        type      zeroGradient;
    }


    topWall
    {
        type      zeroGradient;
    }

    bottomWall
    {
        type      zeroGradient;
    }

    discShaft_2mm
    {
        type      zeroGradient;
    }

    inlet
    {
        type      zeroGradient;
    }


    outlet
    {
        type       fixedValue;
        value      uniform 0;
    }

}

// ************************************************************************* //
```

Listing N.15: Initial and boundary conditions of the turbulent viscosity for the rs-SDR simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include "$FOAM_CASE/Parameters"        // Include the parameters
#include "$FOAM_CASE/Calculations"      // Include the Calculation for the
    parameters

dimensions      [0 2 -1 0 0 0 0];

internalField   uniform $nu;

boundaryField
{
    walls
    {
        type    nutUSpaldingWallFunction;
        value   uniform 0;
    }

    top
    {
        type    nutUSpaldingWallFunction;
        value   uniform 0;
    }

    bottom
    {
        type    nutUSpaldingWallFunction;
        value   uniform 0;
    }


    topWall
    {
        type    nutUSpaldingWallFunction;
        value   uniform 0;
    }

    bottomWall
    {
        type    nutUSpaldingWallFunction;
        value   uniform 0;
    }

    discShaft_2mm
    {
        type    nutUSpaldingWallFunction;
        value   uniform 0;
    }

    inlet
    {
        type    calculated;
        value   uniform 0;
    }

    outlet
    {
        type    calculated;
        value   uniform 0;
    }

}

// ************************************************************************* //
```

N.3 Turbulent Couette flow

Listing N.16: *Allrun* file of turbulent Couette simulation

```
#!/bin/sh
cd ${0%/*} || exit 1           # Run from this directory
. $WM_PROJECT_DIR/bin/tools/RunFunctions    # Source tutorial run functions

####################### The steps to run the simulation #######################

runApplication blockMesh        # Creating the mesh

runApplication checkMesh        # Checking the mesh

runApplication decomposePar -force        # Divide the workload over the cores

runApplication mpirun -np 6 pisoFoam -parallel    # Run the simulation in parallel

runApplication reconstructPar -latestTime   # Reconstruct the decomposed folders

#------------------------------------------------------------------------------
```

Listing N.17: *blockMeshDict* file specifying the geometry of the turbulent Couette simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1806                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include "../Parameters"
#include "../Calculations"

scale   1;

vertices
(
 (0 0 0)
 ($l 0 0)
 ($l $h 0)
 (0 $h 0)
 (0 0 $w)
 ($l 0 $w)
 ($l $h $w)
 (0 $h $w)
);

blocks(

hex (0 1 2 3 4 5 6 7) ($Nx $Ny $Nz) simpleGrading (1  1  1)
    );

edges
(
);


boundary
(
    // Define the boundaries of the 1 block system (easiest way to understand is by
     drawing the system)
    top
    {
      type wall;
      faces
      (
        (3 7 6 2)
      );
    }

    bottom
    {
      type wall;
      faces
      (
        (0 4 5 1)
      );
    }

    inlet
    {
      type cyclic;
      neighbourPatch outlet;
      faces
      (
        (0 3 7 4)
      );
    }
    outlet
    {
      type cyclic;
      neighbourPatch inlet;
      faces
      (
        (1 2 6 5)
      );
    }


    front
    {
      type cyclic;
      neighbourPatch back;
      faces
      (
        (0 3 2 1)
      );
    }

    back
    {
      type cyclic;
      neighbourPatch front;
      faces
      (
        (4 7 6 5)
      );
    }
);

// ************************************************************************* //
```

Listing N.18: *fvSolution* file specifying the solvers of the turbulent Couette simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1806                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    p
    {
        solver                  GAMG;
        smoother                GaussSeidel;
        preconditioner          DIC;
        tolerance               1e-06;
        relTol                  0.05;
        cacheAgglomeration      yes;
        nPreSweeps              2;
        nPostSweeps             2;
    }

    pFinal
    {
        $p;
        relTol          0;
    }

  "(U|nut|nuTilda)"
    {
        solver          smoothSolver;
        smoother        GaussSeidel;
        tolerance       1e-05;
        relTol          0;
    }
}

PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}


// ************************************************************************* //
```

Listing N.19: *fvSchemes* file specifying the schemes of the turbulent Couette simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|   \\    /   O peration      | Version:   v1806                                |
|    \\  /    A nd            | Web:       www.OpenFOAM.com                     |
|     \\/     M anipulation   |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         CrankNicolson 0.9;
}

gradSchemes
{
    default     Gauss cubic;
    grad(p)     Gauss cubic;
    grad(U)     Gauss cubic;
    grad(nut)   Gauss cubic;

}

divSchemes
{
  default                             Gauss linear;
  div(phi,U)                          Gauss linear;
  div(phi,nut)                        Gauss linear;
  div(yPhi,yWall)                     Gauss linear;
  div((nuEff*dev2(T(grad(U)))))       Gauss linear;
}

laplacianSchemes
{
    default           Gauss linear corrected;
    laplacian(yWall)  Gauss linear orthogonal;
}

interpolationSchemes
{
    default     linear;
}

snGradSchemes
{
    default     orthogonal;
}

wallDist
{
    method      meshWave;
}


// ************************************************************************* //
```

Listing N.20: Initial and boundary conditions of the velocity for the turbulent Couette simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1906                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include "../Parameters"  // Include the parameters
#include "../Calculations"  // Include the Calculation for the parameters

dimensions      [0 1 -1 0 0 0 0];


internalField   #codeStream
        {
                codeInclude
                #{
                #include "fvCFD.H"
                #};

                codeOptions
                #{
                -I$(LIB_SRC)/finiteVolume/lnInclude \
                -I$(LIB_SRC)/meshTools/lnInclude
                #};

                codeLibs
                #{
                -lmeshTools \
                -lfiniteVolume
                #};

                code
                #{
                const IOdictionary& d = static_cast<const IOdictionary&>(dict);
                const fvMesh& mesh = refCast<const fvMesh>(d.db());

                vectorField v(mesh.nCells());

                forAll(v, i)
                {
                const scalar y = mesh.C()[i][1];

                scalar vX = y/$h*$v; // In the case of stationary bottom wall

                // Add turbulent perturbations

                scalar v_x_rand = ( rand() * 1e-9 ) - 1.2;
                scalar v_y_rand = ( rand() * 1e-9 ) - 1.2;
                scalar v_z_rand = ( rand() * 1e-9 ) - 1.2;


                scalar vX_perturb = vX + v_x_rand;
                v[i] = vector(vX_perturb,v_y_rand,v_z_rand);

                }

                writeEntry(os, "", v);

                #};
        };

boundaryField
{
    top
    {
        type            fixedValue;
        value           uniform ($v 0 0);
    }

    bottom
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

    inlet
    {
        type            cyclic;
    }

    outlet
    {
        type            cyclic;
    }

    front
    {
        type            cyclic;
    }

    back
    {
        type            cyclic;
    }
}

// ************************************************************************* //
```

Listing N.21: Initial and boundary conditions of the pressure for the turbulent Couette simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1906                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0];


internalField   uniform 0;

boundaryField
{
    top
    {
        type            zeroGradient;

    }

    bottom
    {
        type            zeroGradient;
    }

    inlet
    {
        type            cyclic;
    }

    outlet
    {
        type            cyclic;
    }

    front
    {
        type            cyclic;
    }

    back
    {
        type            cyclic;
    }
}

// ************************************************************************* //
```

Listing N.22: Initial and boundary conditions of the turbulent viscosity for the turbulent Couette simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1906                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      nut;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include "../Parameters"        // Include the parameters

dimensions      [0 2 -1 0 0 0 0];

internalField   uniform $nu;

boundaryField
{
    top
    {
        type            nutUSpaldingWallFunction;
        value           uniform 0;
    }

    bottom
    {
        type            nutUSpaldingWallFunction;
        value           uniform 0;
    }

    inlet
    {
        type            cyclic;
    }

    outlet
    {
        type            cyclic;
    }


    front
    {
        type            cyclic;
    }

    back
    {
        type            cyclic;
    }
}

// ************************************************************************* //
```

N.4 Bacterial-resolved flow

Listing N.23: *Allrun* file of the bacterial-resolved flow simulation

```
#!/bin/sh
cd ${0%/*} || exit 1    # Run from this directory

# Source tutorial run functions
. $WM_PROJECT_DIR/bin/tools/RunFunctions

####################### The steps to run the simulation #######################

runApplication blockMesh # Make the mesh

# Perform gradual refining
for i in 1 2 3 4 5 6 7 8 9
do
    runApplication -s $i \
        topoSet -dict system/topoSetDict.${i} -time 0

    runApplication -s $i \
        refineMesh -overwrite -dict system/refineMeshDict.${i}
done

runApplication checkMesh # Check the mesh

# Clean the folder
rm -r 0/
cp -r 0.orig/ 0/

# Map the flowfield of a Couette flow

runApplication mapFields ../Flowfield_21333 -consistent -sourceTime latestTime
runApplication funkySetFields -time 0  # Insert the bacterium
runApplication decomposePar -force -latestTime # Decomposing
runApplication mpirun -np 6 multiphaseInterFoam -parallel # Running in parallel
runApplication reconstructParMesh  # Reconstruct the mesh
runApplication reconstructPar # Reconstruct the fields
#----------------------------------------------------------------------
```

Listing N.24: *fvSolution* file specifying the solvers of the bacterial-resolved flow simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    "alpha.*"
    {
        nAlphaCorr      1;
        nAlphaSubCycles 2;
        cAlpha      1;
        cycleAlpha  yes;
        solver    smoothSolver;
        smoother  symGaussSeidel;
        tolerance 1e-8;
        relTol    0;
    }

    "pcorr"
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-10;
        relTol          0;
    }

   pcorrFinal
   {
        $pcorr;
        relTol        0;
   }

    p_rgh
    {
        solver          GAMG;
        smoother        DIC;
        tolerance       1e-8;
        relTol          0;
    }

    p_rghFinal
    {
        $p_rgh;
        relTol          0;
    }

    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-5;
        relTol          0;
        minIter         1;
    }

    UFinal
    {
        $U;
        relTol          0;
    }

    "e.*"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-8;
        relTol          0;
        minIter         1;
    }

    "(k|epsilon|Theta).*"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-7;
        relTol          0;
        minIter         1;
    }
}

PIMPLE
{
  momentumPredictor   yes;
  nOuterCorrectors    1;
  nCorrectors         2;
  nNonOrthogonalCorrectors 4;
  pRefCell        0;
  pRefValue       0;
  turbOnFinalIterOnly   no;
}
```

```

relaxationFactors
{
    fields
    {
       p         0.3;
       p_corr    0.3;
    }

    equations
    {
       U       0.6;
    }
}


// ************************************************************************* //
```

Listing N.25: *fvSchemes* file specifying the schemes of the bacterial-resolved flow simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         CrankNicolson 1;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default                                 Gauss linear;
    div(rhoPhi,U)                           Gauss linearUpwind grad(U);
    div(rho*phi,U)                          Gauss limitedLinearV 1;
    div(phi,alpha)                          Gauss MPLICU;
    div(phirb,alpha)                        Gauss linear;
    div(((rho*nuEff)*dev2(T(grad(U)))))     Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

```

```

// ************************************************************************* //
```

Listing N.26: *dynamicMeshDict* file specifying the moving mesh of the bacterial-resolved flow simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      dynamicMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#include "../Parameters"
#include "../Calculations"

dynamicFvMesh dynamicMotionSolverFvMesh;

motionSolverLibs ("libfvMotionSolvers.so");

motionSolver solidBody;

cellZone none;

solidBodyMotionFunction linearMotion;

velocity  ($v_mesh 0.0 0.0);


// ************************************************************************* //
```

Listing N.27: *funkySetFieldsDict* file specifying the initialisation of the bacterium in the bacterial-resolved flow simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      funkySetFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#include "../Parameters"
#include "../Calculations"

expressions
(
  InitFieldAlpha
  {
    field alpha.bac;
    variables
    (
    "h=1e-3;"
    "l=0.2*h;"
    "w=1*h;"
    "xc=0.5*l;"
    "yc=0.5*h;"
    "zc=0.5*w;"
    "rbac=0.25e-6;"
    "lbac=3.2e-6;"
    "k=0.9;"
    );
    expression "1";
    condition "(sqr(pos().x-xc)/sqr(rbac)+sqr(pos().y-(yc-0.5*k*lbac))/sqr((1-k)*
    lbac)+sqr(pos().z-zc)/sqr(rbac)<1) || (sqr(pos().x-xc)/sqr(rbac)+sqr(pos().y-(
    yc+0.5*k*lbac))/sqr((1-k)*lbac)+sqr(pos().z-zc)/sqr(rbac)<1) || ((pos().y > (yc
    -0.5*k*lbac)) && (pos().y < (yc+0.5*k*lbac)) && (sqr(pos().x-xc)+sqr(pos().z-zc
    ) < sqr(rbac)))";
    keepPatches true;
  }
  InitFieldAlpha2
  {
    field   alpha.water;
    variables
    (
    "h=1e-3;"
    "l=0.2*h;"
    "w=1*h;"
    "xc=0.5*l;"
    "yc=0.5*h;"
    "zc=0.5*w;"
    "rbac=0.25e-6;"
    "lbac=3.2e-6;"
    "k=0.9;"
    );
    expression "0";
    condition "(sqr(pos().x-xc)/sqr(rbac)+sqr(pos().y-(yc-0.5*k*lbac))/sqr((1-k)*
    lbac)+sqr(pos().z-zc)/sqr(rbac)<1) || (sqr(pos().x-xc)/sqr(rbac)+sqr(pos().y-(
    yc+0.5*k*lbac))/sqr((1-k)*lbac)+sqr(pos().z-zc)/sqr(rbac)<1) || ((pos().y > (yc
    -0.5*k*lbac)) && (pos().y < (yc+0.5*k*lbac)) && (sqr(pos().x-xc)+sqr(pos().z-zc
    ) < sqr(rbac)))";
    keepPatches true;
  }


  // Filter the turbulent fluctuation near bacterium in initialization
  VelOverall
  {
    field U;
    variables
    (
    "Re=21333;"
    "nu_water=1e-6;"
    "h=1e-3;"
    "v=Re*nu_water/(0.25*h);"
    );
    expression "vector((v/h)*(pos().y),0,0)";
    condition "(pos().y < 0.52 * h) && (pos().y > 0.48 * h) ";
    keepPatches true;
  }

);

// ************************************************************************* //
```

Listing N.28: *phaseProperties* file specifying the phases in the bacterial-resolved flow simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
```

```
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      phaseProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

type    basicMultiphaseSystem;

phases (bac water);

bac
{
    type            purePhaseModel;
    diameterModel   isothermal;
    isothermalCoeffs
    {
        d0              3e-6;
        p0              1e5;
    }

    residualAlpha   1e-6;
}

water
{
    type            purePhaseModel;
    diameterModel   constant;
    constantCoeffs
    {
        d               1e-4;
    }

    residualAlpha   1e-6;
}

blending
{
    default
    {
        type            linear;
    // minimum volume fraction of a phase to be considered as continuous phase
        minFullyContinuousAlpha.bac 1;
        minPartlyContinuousAlpha.bac 0;
        minFullyContinuousAlpha.water 1;
        minPartlyContinuousAlpha.water 0;
    }

    drag
    {
        type            linear;
        minFullyContinuousAlpha.bac 0.7;
        minPartlyContinuousAlpha.bac 0.5;
        minFullyContinuousAlpha.water 0.7;
        minPartlyContinuousAlpha.water 0.5;
    }
}

surfaceTension
(
    (bac and water)
    {
        type            constant;
        sigma   0.056;     //[N/m]
    }
);

aspectRatio
(
    (bac in water)
    {
        type            constant;
        E0              6.4;
    }

    (water in bac)
    {
        type            constant;
        E0              1.0;
    }
);

drag
(
    (bac in water)
    {
        type            SchillerNaumann;
        residualRe      1e-3;
        swarmCorrection
        {
            type        none;
        }
    }

    (water in bac)
    {
        type            SchillerNaumann;
        residualRe      1e-3;
        swarmCorrection
        {
            type        none;
        }
    }

    (bac and water)
    {
        type            segregated;
        m               0.5;
        n               8;
        swarmCorrection
        {
            type        none;
        }
    }
);

virtualMass
(
    (bac in water)
    {
        type            constantCoefficient;
        Cvm             0.5;
    }

    (water in bac)
    {
        type            constantCoefficient;
        Cvm             0.5;
    }
);

heatTransfer
(
    (bac in water)
    {
        type            RanzMarshall;
        residualAlpha   1e-4;
    }

    (water in bac)
    {
        type            RanzMarshall;
        residualAlpha   1e-4;
    }
```

```
);

phaseTransfer
();

lift
();

wallLubrication
();

turbulentDispersion
();

interfaceCompression
();

// Minimum allowable pressure
pMin            10000;


// ************************************************************************* //
```

Listing N.29: *transportProperties* file specifying the momentum transport in the bacterial-resolved flow simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  8
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

phases
(
     water
     {
         transportModel  Newtonian;
         nu   [ 0 2 -1 0 0 0 0 ] 1e-06;
         rho  [ 1 -3 0 0 0 0 0 ] 997;
     }

     bac
     {
         transportModel  HerschelBulkley;
         nu0    [ 0 2 -1 0 0 0 0 ] 6.55e04;
         tau0   [ 0 2 -2 0 0 0 0 ] 2.73e03;
         k      [ 0 2 -1 0 0 0 0 ] 0.0909;
         n      [ 0 0  0 0 0 0 0 ] 1;
         rho    [ 1 -3 0 0 0 0 0 ] 1100;

     }

);

sigmas
(
    (water bac) 0.056
);


// ************************************************************************* //
```

N.5 MP-PIC simulation

Listing N.30: *Allrun* file of the MP-PIC simulation

```
#!/bin/sh
cd ${0%/*} || exit 1    # Run from this directory

# Source tutorial run functions
. $WM_PROJECT_DIR/bin/tools/RunFunctions

rm -rf 0/
cp -r 0.orig/ 0/

runApplication blockMesh

runApplication surfaceFeatures                    # Extract edges from trisurface

runApplication snappyHexMesh -overwrite

runApplication topoSet                                  # Define additional faces

runApplication createPatch -overwrite                   # Make additional patches

runApplication checkMesh  #Check the mesh

mv 0/U.water 0/UMean
mv 0/nut.water 0/nut

# Map the mean results of the simulation to the geometry
runApplication mapFields ../rsSDR_throughflow_2mm -sourceTime latestTime

mv 0/UMean 0/U.water
mv 0/nut 0/nut.water

runApplication decomposePar -force

runApplication mpirun -np 256 denseParticleFoam -parallel   # Run the simulation in
     parallel

runApplication reconstructPar -latestTime   # Reconstruct the decomposed folders

#------------------------------------------------------------------------------
```

Listing N.31: *fvSolution* file specifying the solvers of the MP-PIC simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  9
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    alpha.water
    {
        max             1;
    }

    p
    {
        solver          GAMG;
        tolerance       1e-06;
        relTol          0.01;
        smoother        GaussSeidel;
    }

    pFinal
    {
        solver          GAMG;
        tolerance       1e-06;
        relTol          0;
        smoother        GaussSeidel;
    }

    "(U|k|epsilon|omega).water"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
        relTol          0.1;
    }

    "(U|k|epsilon|omega).waterFinal"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
        relTol          0.1;
    }

    cloud:alpha
    {
        solver          GAMG;
        tolerance       1e-06;
        relTol          0.1;
        smoother        GaussSeidel;
    }
}

PIMPLE
{
    nOuterCorrectors 1;
    nCorrectors     2;
    momentumPredictor yes;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}

relaxationFactors
{
}


// ************************************************************************* //
```

Listing N.32: *fvSchemes* file specifying the schemes of the MP-PIC simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  9
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    format      ascii;
    class       dictionary;
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         CrankNicolson 0.9;
}

gradSchemes
{
    default         Gauss cubic;
}

divSchemes
{
    default                               none;
    div(alphaPhi.water,U.water)           Gauss linearUpwindV unlimited;
    div(((alpha.water*nuEff.water)*dev2(T(grad(U.water))))) Gauss linear;
    div(phiGByA,cloud:alpha)              Gauss linear;
    div(alphaPhi.water,epsilon.water)     Gauss limitedLinear 1;
    div(alphaPhi.water,k.water)           Gauss limitedLinear 1;
}

laplacianSchemes
{
    default         Gauss cubic corrected;
}

interpolationSchemes
{
    default         cubic;
}

snGradSchemes
{
    default         corrected;
}

// ************************************************************************* //
```

Listing N.33: *cloudProperties* file specifying the particle dynamics properties in the MP-PIC simulation

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  9
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    format      ascii;
    class       dictionary;
    location    "constant";
    object      cloudProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include "$FOAM_CASE/Parameters"        // Include the parameters
#include "$FOAM_CASE/Calculations"      // Include the Calculation for the
    parameters

type        MPPICCloud;

solution
{
    coupled         true;
    transient       yes;
    cellValueSourceCorrection off;
    maxCo           0.3;

    interpolationSchemes
    {
        rho.water          cell;
        U.water            cellPoint;
        mu.water           cell;
        alpha.water        cell;
        curlUcDt           cellPoint;
        DUcDt             cellPoint;
    }

    averagingMethod dual;

    integrationSchemes
    {
        U               Euler;
    }

    sourceTerms
    {
        schemes
        {
            U           semiImplicit 1;
        }
    }
}

constantProperties
{
    rho0            $rho_bac;
}

subModels
{
    particleForces
    {
    nonSphereDrag
    {
      alphac  alpha.water;
      phi     0.6546;
    }

    gravity;

    virtualMass
    {
      Cvm   0.5;
      U     U.water;
    }

    pressureGradient
    {
      U     U.water;
    }

    SaffmanMeiLiftForce
    {
      U     U.water;
    }
    }

    injectionModels
    {
        model1
        {
            type            patchInjection;
            massTotal       $mass_inj;
            SOI             0;
            parcelBasisType mass;
            patchName       inlet;
            duration        $inj_time;
            parcelsPerSecond 1e6;
            U0              (0 0 $v_in);
            flowRateProfile constant 1;
            sizeDistribution
            {
                type        fixedValue;
                fixedValueDistribution
                {
                    value $l_bac;
                }
            }
        }
    }

    dispersionModel none;

    patchInteractionModel localInteraction;

    localInteractionCoeffs
    {
        patches
        (
            walls
            {
                type rebound;
                e    0.97;
                mu   0.09;
            }
            inlet
            {
                type rebound;
                e    0.97;
                mu   0.09;
            }
            topWall
            {
                type rebound;
                e    0.97;
                mu   0.09;
            }
            bottomWall
            {
                type rebound;
                e    0.97;
                mu   0.09;
            }
            discShaft_2mm
            {
                type rebound;
                e    0.97;
                mu   0.09;
            }
            outlet
            {
                type escape;
            }
        );
    }

    heatTransferModel none;

    surfaceFilmModel none;

    packingModel implicit;

    implicitCoeffs
    {
        alphaMin 1e-6;
        rhoMin 1.0;
        applyLimiting true;
        applyGravity false;
        particleStressModel
        {
            type HarrisCrighton;
            alphaPacked 0.4;
            pSolid 5.0;
            beta 2.0;
            eps 1.0e-2;
        }
    }

    dampingModel none;

    isotropyModel none;

    stochasticCollisionModel none;

    radiation off;
}


cloudFunctions
{}


// ************************************************************************* //
```

N.6 RTD solver

Listing N.34: *createFields* file extracting the fields to be used to predict the RTD

```
Info<< "Reading field T\n" << endl;

volScalarField T
(
    IOobject
    (
        "T",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);


Info<< "Reading field U\n" << endl;

volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);


Info<< "Reading field nut\n" << endl;

volScalarField nut
(
    IOobject
    (
        "nut",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);


Info<< "Reading transportProperties\n" << endl;

IOdictionary transportProperties
(
    IOobject
    (
        "transportProperties",
        runTime.constant(),
        mesh,
        IOobject::MUST_READ_IF_MODIFIED,
        IOobject::NO_WRITE
    )
);

Info<< "Reading viscosity nu\n" << endl;

dimensionedScalar nu
(
     transportProperties.lookup("nu")
);


Info<< "Reading molecular diffusivity Dm\n" << endl;

dimensionedScalar Dm
(
    transportProperties.lookup("Dm")
);

Info<< "Reading turbulent Schmidt number ScT\n" << endl;

dimensionedScalar ScT
(
    transportProperties.lookup("ScT")
);

volScalarField Dt
(
  IOobject
        (
            "Dt",
            runTime.timeName(),
            mesh,
            IOobject::NO_READ,
            IOobject::NO_WRITE
        ),
        nut / ScT + Dm
);

#include "createPhi.H"

#include "createFvOptions.H"
```

Listing N.35: *RTDFoam* file specifying the scalar transport solver to be used to predict the RTD

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Copyright (C) 2011-2018 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.
```

```

Application
    RTDFoam

Description
    Solves the transient transport equation for a scalar on a
    turbulent field for RTD purposes.

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "fvOptions.H"
#include "simpleControl.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
    #include "setRootCaseLists.H"
    #include "createTime.H"
    #include "createMesh.H"

    simpleControl simple(mesh);

    #include "createFields.H"

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nCalculating scalar transport\n" << endl;

    #include "CourantNo.H"

    while (simple.loop(runTime))
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

        while (simple.correctNonOrthogonal())
        {
            fvScalarMatrix TEqn
            (
                fvm::ddt(T)
              + fvm::div(phi, T)
              - fvm::laplacian(Dt, T)

            );

            TEqn.relax();
            fvOptions.constrain(TEqn);
            TEqn.solve();
            fvOptions.correct(T);

  }

        runTime.write();
    }

    Info<< "End\n" << endl;

    return 0;
}


// ************************************************************************* //
```

N.7 Data analysis rotor-stator cavity

Listing N.36: Data analysis script for the rotor-stator cavity simulations

```
%This file extracts the velocity data from the OpenFOAM
%simulation singleGraph postProcessing utility

```

```
%%% Validation turbulence properties %%%%
clear; close all; clc;
%% Simulation constants
%%%% User input desired simulation %%%%
% See Excel file for details simulations
pSG             =      1000;                      %Number of points in singleGraphs

simN            =      'rsCav_7';           %Name simulation

% insert the latest time in string format that is in the postProcessing folder

Time = "1.000";

% Axial coordinates at which the data is plotted
list_ax_pos = [0.01 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 0.99];

%% Extract Data

for i= 11:21
    k                   =       i-10;
    sGN                 =       strcat('graphUniform',num2str(i));

    data_scalars = table2array(readtable(fullfile(simN,'postProcessing',sGN,Time,'
    line_scalars')));
    data_scalars = unique(data_scalars,'rows');

    data_symmtensors = table2array(readtable(fullfile(simN,'postProcessing',sGN,
    Time,'line_UPrime2Mean_R_RMean')));
    data_symmtensors = unique(data_symmtensors,'rows');

    % Make a matrix of pSG long, with at each pSG the [3x3] gradUPrimeSqrMean
    tensor
    gUP2Mean = zeros(pSG,3,3);

    for j = 1:pSG
    gUP2Mean(j,1,:) = data_scalars(j,9:11);
    gUP2Mean(j,2,:) = data_scalars(j,12:14);
    gUP2Mean(j,3,:) = data_scalars(j,15:17);
    end

    % Make a matrix of pSG long, with at each pSG the [3x3] gradUPrimeMean tensor
    gUPMean = zeros(pSG,3,3);

    for j = 1:pSG
    gUPMean(j,1,:) = data_scalars(j,18:20);
    gUPMean(j,2,:) = data_scalars(j,21:23);
    gUPMean(j,3,:) = data_scalars(j,24:26);
    end

  % Compute the covariances and correct the averages
    covariance_xy = cov(gUPMean(:,1,2),gUPMean(:,2,1));
    crossProdMean_xy = covariance_xy(1,2) + gUPMean(:,1,2).*gUPMean(:,2,1);

    covariance_xz = cov(gUPMean(:,1,3),gUPMean(:,3,1));
    crossProdMean_xz = covariance_xz(1,2) + gUPMean(:,1,3).*gUPMean(:,3,1);

    covariance_yz = cov(gUPMean(:,2,3),gUPMean(:,3,2));
    crossProdMean_yz = covariance_yz(1,2) + gUPMean(:,2,3).*gUPMean(:,3,2);

  % Extract the effective viscosity
    nuEffMean = data_scalars(:,6);

  % Extract the sub-grid-scale energy dissipation rate
    eps_SGS = data_scalars(:,3);

    for j = 1:pSG
    EDR_resolved(j,1) = nuEffMean(j) * (2 * gUP2Mean(j,1,1) + gUP2Mean(j,2,1) +
    gUP2Mean(j,3,1) + gUP2Mean(j,1,2) + ...
                             2 * gUP2Mean(j,2,2) + gUP2Mean(j,3,2) + gUP2Mean(j
    ,1,3) + gUP2Mean(j,2,3) + ...
                             2* gUP2Mean(j,3,3) + 2*( crossProdMean_xy(j) +
    crossProdMean_xz(j) + ...
                             crossProdMean_yz(j)));
    end

  % Compute the total energy dissipation rate
    EDR_total_LES(:,k) = eps_SGS + EDR_resolved;
    rad_pos_LES(:,k)   = abs(data_scalars(:,1));

    UPrime2Mean = data_symmtensors(:,2:8);
    k_unresolved = data_scalars(:,2);

  % Compute the total turbulent kinetic energy
    TKE(:,k) = 0.5.*(UPrime2Mean(:,1)+UPrime2Mean(:,4)+UPrime2Mean(:,6)) +
    k_unresolved;
end

%% plotting

% plot EDR as a function of radial position for each axial position

figure(1)
loglog(rad_pos_LES(2:end,:),EDR_total_LES(2:end,:))
xlabel('Radial position [m]')
ylabel('Energy dissipation rate [w/kg]')
legendCell = strcat(string(num2cell(list_ax_pos)),'h');
legend(legendCell)
title('Radial profiles of the EDR for LES')

% plot mean EDR as a function of axial position

for i = 1:length(list_ax_pos)
    EDR_total_LES_RadialAvg(i) = mean(EDR_total_LES(:,i));
end

figure(2)
plot(EDR_total_LES_RadialAvg,list_ax_pos)
ylabel('z^* [-]')
xlabel('Energy dissipation rate [W/kg]')
title('Radially averaged EDR for LES')

% plot the mean energy dissipation rate as a function of the radial
% position
EDR_total_LES_AxialAvg = trapz(list_ax_pos,EDR_total_LES,2)./(list_ax_pos(end)-
    list_ax_pos(1));
figure(3)
loglog(rad_pos_LES,EDR_total_LES_AxialAvg)
xlabel('Radial position [m]')
ylabel('Energy dissipation rate [w/kg]')
title('Axially averaged EDR for LES')

% Check the increment of EDR as a function of r
for i = 1:length(EDR_total_LES_AxialAvg)-1
    slope_LES(i) = (log(EDR_total_LES_AxialAvg(i+1))-log(EDR_total_LES_AxialAvg(i))
    )/...
        (log(rad_pos_LES(i+1))-log(rad_pos_LES(i)));
end

figure(4)
plot(rad_pos_LES(1:end-1,1),movmean(slope_LES,100))
xlabel('Radial position [m]')
ylabel('Slope [-]')
title('Slope in loglog plot for LES')
mean_slope_LES = mean(slope_LES)

%% Different Axial profile: Extract Data

    sGN                 =       'graphUniform24';

    data_scalars = table2array(readtable(fullfile(simN,'postProcessing',sGN,Time,'
    line_scalars')));
    data_scalars = unique(data_scalars,'rows');

    data_symmtensors = table2array(readtable(fullfile(simN,'postProcessing',sGN,
    Time,'line_UPrime2Mean_R_RMean')));
    data_symmtensors = unique(data_symmtensors,'rows');

    % Make a matrix of pSG long, with at each pSG the [3x3] gradUPrimeSqrMean tensor
    gUP2Mean = zeros(pSG,3,3);

    for j = 1:pSG
    gUP2Mean(j,1,:) = data_scalars(j,9:11);
    gUP2Mean(j,2,:) = data_scalars(j,12:14);
    gUP2Mean(j,3,:) = data_scalars(j,15:17);
    end

    % Make a matrix of pSG long, with at each pSG the [3x3] gradUPrimeMean tensor
    gUPMean = zeros(pSG,3,3);

    for j = 1:pSG
    gUPMean(j,1,:) = data_scalars(j,18:20);
    gUPMean(j,2,:) = data_scalars(j,21:23);
    gUPMean(j,3,:) = data_scalars(j,24:26);
    end

  % Compute the covariances and correct the averages
    covariance_xy = cov(gUPMean(:,1,2),gUPMean(:,2,1));
    crossProdMean_xy = covariance_xy(1,2) + gUPMean(:,1,2).*gUPMean(:,2,1);

    covariance_xz = cov(gUPMean(:,1,3),gUPMean(:,3,1));
    crossProdMean_xz = covariance_xz(1,2) + gUPMean(:,1,3).*gUPMean(:,3,1);

    covariance_yz = cov(gUPMean(:,2,3),gUPMean(:,3,2));
    crossProdMean_yz = covariance_yz(1,2) + gUPMean(:,2,3).*gUPMean(:,3,2);

  % Extract the effective viscosity
    nuEffMean = data_scalars(:,6);

  % Extract the sub-grid-scale energy dissipation rate
    eps_SGS = data_scalars(:,3);

    for j = 1:pSG
    EDR_resolved(j,1) = nuEffMean(j) * (2 * gUP2Mean(j,1,1) + gUP2Mean(j,2,1) + gUP2Mean(j,3,1) + gUP2Mean(j,1,2) + ...
                                 2 * gUP2Mean(j,2,2) + gUP2Mean(j,3,2) + gUP2Mean(j,1,3) + gUP2Mean(j,2,3) + ...
                                 2* gUP2Mean(j,3,3) + 2*( crossProdMean_xy(j) + crossProdMean_xz(j) + ...
                                 crossProdMean_yz(j)));
    end

    axial_pos_graphUniform24(:)   = abs(data_scalars(:,1));

    UPrime2Mean = data_symmtensors(:,2:8);
    k_unresolved = data_scalars(:,2);

  % Compute the total turbulent kinetic energy
    TKE_graphUniform24 = 0.5.*(UPrime2Mean(:,1)+UPrime2Mean(:,4)+UPrime2Mean(:,6)) + k_unresolved;
  % Compute the total energy dissipation rate
    EDR_total_graphUniform24 = eps_SGS + EDR_resolved;

%% plotting
figure(5)
semilogx(EDR_total_graphUniform24,axial_pos_graphUniform24./0.009)
ylabel('z^* [-]')
xlabel('Energy dissipation rate [W/kg]')
title('EDR for LES')

figure(6)
plot(TKE_graphUniform24,axial_pos_graphUniform24./0.009)
ylabel('z^* [-]')
xlabel('Turbulent kinetic energy [m^2 s^{-2}]')
title('TKE for LES')
```

N.8 Data analysis RTD experiments

Listing N.37: Data analysis script for the RTD experiments

```
% This script is used to find the RTD curves of a rs-SDR
...configuration based on the in- and outlet UV VIS signal
clear; clc; close all;
%% Constants
%Extinction coefficients methylene blue UV-VIS spectrum [m^2/mol]

extCoeff_in613  =          476.5;          %664 nm peak inlet
extCoeff_out613 =          521.7;          %664 nm peak outlet

%Path length UV-VIS cells [mm]
lIn             =          10;                %Inlet
lOut            =          10;                %Outlet

%rs-SDR configuration
h1              =          2e-3;           %Gap distance bottom[m]
h2              =          2e-3;           %Gap distance top [m]
rD              =          0.066;            %Radius disk [m]
rR              =          0.067;            %Radius SDR [m]
rI              =          0.006;            %Inner radius [m]
hD              =          0.008;            %Thicknes rotor [m]

G               =          h1/rD;            %Axial clearance
Vr              =          (h1+h2+hD)*pi*rR^2-hD*pi*rD^2-...
                           h1*pi*rI^2;  %Reactor volume [m3]

nu              =          1e-6;             %Viscosity [m^2/s^2]

% Setting on the pump
phi_set           =          22;
% Volumetric flowrate pump from calibration [m3/s]
phi               =          4.4744e-07*phi_set;

% List of runs on different RPMs
List_RPMs = [71; 95; 119; 143; 239; 334; 382; 429; 477; 525; 573; 621; 716; 859;
    1002; 1145];

%% Start loop over all data
for j = 1:length(List_RPMs)
expN = cell2mat(strcat({'runs '},num2str(List_RPMs(j)),{'_22'}));

for k = 1:3
dataSetN = cell2mat(strcat(num2str(List_RPMs(j)),{'_22 '},num2str(k)));

% Tested rotational rate [RPM] (user input)
rRate0          =          List_RPMs(j);
w               =          rRate0*2*pi/60;   %[Rad/s]
Re              =          w*rD^2/nu;        %Reynolds number

% Import UV-VIS data in- and outlet

dataUV_in613    =        readtable(fullfile('Experimental data',expN,dataSetN,'
    inlet613nm.dat'),...
                         'HeaderLines',9);
dataUV_out613   =        readtable(fullfile('Experimental data',expN,dataSetN,'
    outlet613nm.dat'),...
                         'HeaderLines',9);

%% Construct dataset
% Choose UV-VIS signal (sensitivity versus range of operation)
UV_in           =        dataUV_in613.Var3;
UV_out          =        dataUV_out613.Var3;

extCoeff_in     =        extCoeff_in613;
extCoeff_out    =        extCoeff_out613;

%Date time vectors [clock time]
tUV_in          =        dataUV_in613.Var1;
tUV_out         =        dataUV_out613.Var1;
```

```

%Time vectors [s]
sUV_in          =       dataUV_in613.Var2;
sUV_out         =       dataUV_out613.Var2;

%Dark measurement reference
n0_in           =       -4.7;                           %Inlet
n0_out          =       -3.41;                          %Outlet

%Light measurment reference
nRef_in         =       61802.31;                       %Inlet
nRef_out        =       33508.53;                       %Outlet

%% Clean data
%Set data lower than threshold to value surrounding points
...to filter out disturbances caused by vibrations/air bubbles
Ithreshold       =       0.5;
nShift          =       10;
indexDis        =       UV_out<nRef_out*Ithreshold;
indexRem        =       false(length(indexDis),1);
indexRem(nShift+1:end)  =   indexDis(1:end-nShift);
UV_out(indexDis)     =   UV_out(indexRem);

%% Find injections
%Detect significant change from baseline inlet UV

threshold        =       0.6;     %Threshold fraction reference
                                ...intensity denoting injection
tRecord_b       =       1;       %Time to record before injection [s]
tRecord_in      =       4;       %Time to record after injection
                                ...input signal [s]
tRecord_out     =       50;     %Time to record output signal [s]
nRecord         =       100;     %Points to record

% Time vector [s]
dt              =       mean([sUV_out(end)/length(sUV_out) ...
                        sUV_in(end)/length(sUV_in)]);
nData           =       ceil((tRecord_in+tRecord_out)/dt);
t               =       0:dt:(tRecord_in+tRecord_out);

% Find significant change from base line
index           =       UV_in/nRef_in<threshold;
index           =       [0 ; index(2:end)-index(1:end-1)];

timeInj         =       tUV_in(index>0);     %Injection time [s]

% Ensure that only the initial injection time is selected
timeInj = unique(timeInj,'rows');
%timeInj = timeInj(1);
timeInj = timeInj(end);

%% Calculate RTD curve
% Loop over every injection to find RTD at certain rotational rate
...and flowrate

%Start time [s]
timeS           =       timeInj-tRecord_b/3600/24;
%End time inlet signal [s]
timeE_in        =       timeInj+tRecord_in/3600/24;
%End time outlet signal [s]
timeE_out       =       timeInj+(tRecord_out+tRecord_b)...
                            /3600/24;

indexS_in       =       find(tUV_in==timeS, 1,'first');
indexE_in       =       find(tUV_in==timeE_in, 1,'last');

indexS_out      =       find(tUV_out==timeS, 1,'first');
indexE_out      =       find(tUV_out==timeE_out, 1,'last');

% Shift time if start time does not exist
    tShift          =       1;
    while isempty(indexE_in) || isempty(indexE_out) || ...
        isempty(indexS_in) ||  isempty(indexS_out)

        timeS           =       timeS-tShift/3600/24;
        timeE_in        =       timeE_in-tShift/3600/24;
        timeE_out       =       timeE_out-tShift/3600/24;

        indexS_in       =       find(tUV_in==timeS, 1,'first');
        indexE_in       =       find(tUV_in==timeE_in, 1,'last');

        indexS_out      =       find(tUV_out==timeS, 1,'first');
        indexE_out      =       find(tUV_out==timeE_out, 1,'last');
    end

%Light measurment reference (correct for change in baseline)
nRef_in         =       mean(UV_in(indexS_in-nRecord:...
                            indexS_in));
nRef_out        =       mean(UV_out(indexS_out-nRecord:...
                            indexS_out));

%Absorance
UVAds_in        =       -log10((UV_in(indexS_in:indexE_in)...
                            -n0_in)./(nRef_in-n0_in));
UVAds_out       =       -log10((UV_out(indexS_out:indexE_out)...
                             -n0_out)./(nRef_out-n0_out));

%Concentration [M]
cIn             =       UVAds_in/lIn/extCoeff_in*1e3;
cOut            =       UVAds_out/lOut/extCoeff_out*1e3;

%Normalized concentration
cIn_            =       cIn/trapz(dt,cIn);
cOut_           =       cOut/trapz(dt,cOut);

%Expected mean residence time [s]
tau             =       Vr/phi;

%Find E curve using deconvolution in time domain
[tE , E]        =       deConv(cIn_,cOut_,0,0,dt);

%Deconvolution using fitting of engineering model
diff            =       0.4;
[tE , Efit , p] =       deConvFit(cIn_,cOut_,0,0,dt,tau,diff);

% Convolute RTD with inlet concentration
[tC , cOut_fit] =       conv(cIn_,Efit*dt,0,0,dt);

% Fit parameters (n tanks, tauPFR , tau)
n_tanks_fit         =       p(1);
tauPFR_fit          =       p(2);
tau_mean_fit        =       p(3);

%Save and set data to same base
cOut_           =       interp1(0:dt:(length(cOut_)-1)*dt...
                            ,cOut_,t);
cIn_            =       interp1(0:dt:(length(cIn_)-1)*dt...
                            ,cIn_,t);
E               =       interp1(0:dt:(length(E)-1)*dt...
                            ,E/dt,t);
Efit            =       interp1(0:dt:(length(Efit)-1)*dt...
                            ,Efit,t);
cOut_fit        =       interp1(0:dt:(length(cOut_fit)-1)*dt...
                            ,cOut_fit,t);

%Remove NaN values
cOut_(isnan(cOut_)) =       0;
cIn_(isnan(cIn_))   =       0;
E(isnan(E))         =       0;
Efit(isnan(Efit))   =       0;
cOut_fit(isnan(cOut_fit)) = 0;

%Transition radius from plug flow to well mixed (dimensionless)
%Corrected for fitted residence time ...
%(air occupying part of volume/dead volume)

```

```
rTrans                  =       sqrt((p(2)*phi-(Vr-(h1+h2)*pi*...
                                    rD^2))/(2*pi*rD^2*h1*p(3)...
                                    /tau)+(rI/rD)^2);

%Expected rTransition from theory
cW                      =       phi/nu/rD;
c                       =       0.219;
rTransM                 =       (1/c*cW/Re^(4/5))^(5/13);

%Determine coeffcient of determination (R^2) for fit
cOut_mean               =       mean(cOut_);
SStot                   =       sum((cOut_-cOut_mean).^2);
SSres                   =       sum((cOut_-cOut_fit).^2);
R2                      =       1-SSres/SStot;

n_tanks_fit             =       p(1);
tauPFR_fit              =       p(2);
tau_mean_fit            =       p(3);
V_PFR_norm_fit          =       p(2)/p(3);

%Write to file
save(dataSetN,'n_tanks_fit','tauPFR_fit','tau_mean_fit','V_PFR_norm_fit','E','Efit'
    ,'cOut_','cOut_fit','cIn_','t','R2','rTrans','rTransM');

end

end


%% (De)convolution functions
%Convolution
function [tC , cOut] = conv(cIn,E,tS_in,ts_E,dt)
%Construct Toeplitz matrix
m                   =       length(cIn);
n                   =       length(E);
v                   =       m+n-1;

A                   =       zeros(v,m);
for i=1:m
    A(i:n+i-1,i)=       E;
end
%Perfom matrix multiplication to find outlet concentration
cOut                =       A*cIn;
%Time vector corresponding the outlet concentration
tC                  =       (tS_in+ts_E):dt:(v-1)*dt+(tS_in+ts_E);
end

%Deconvolution
function [tE , E] = deConv(cIn,cOut,tS_in,tS_out,dt)
% Construct time vector
m                   =       length(cIn);
v                   =       length(cOut);
n                   =       v-m+1;

tE                  =       (tS_out-tS_in):dt:(dt*(n-1)+(tS_out-tS_in));

% Find E curve with optimisation
g                   =       @(f) OF(f,cIn,cOut);

E0                  =       ones(n,1); % Initial guess

E                   =       lsqnonlin(g,E0,zeros(n,1),[]);

end

%Objective function
function error  = OF(f,cIn,cOut)
%Construct Toeplitz matrix
m                   =       length(cIn);
v                   =       length(cOut);
n                   =       v-m+1;

A                   =       zeros(v,m);
```

```
for i=1:m
    A(i:n+i-1,i)=       f;
end

error           =       cOut-A*cIn;
end

%Deconvolution by fitting of engineering model
function [tE , E , p] = deConvFit(cIn,cOut,tS_in,tS_out,dt,tau,diff)
%Construct time vector
m               =       length(cIn);
v               =       length(cOut);
n               =       v-m+1;

tE              =       (tS_out-tS_in):dt:(dt*(n-1)+(tS_out-tS_in));

% Find E curve with optimisation
p0              =       [2 2 tau]; % Initial guess

g               =       @(p) OFFit(p,cIn,cOut,tE,dt);

p               =       lsqnonlin(g,p0,[1.001 0 (1-diff)*tau],...
                        [3 inf (1+diff)*tau]);

E               =       EM(tE,p(3),p(1),p(2));

end

%Objective function
function error  = OFFit(p,cIn,cOut,tE,dt)
%Construct Toeplitz matrix
m               =       length(cIn);
v               =       length(cOut);
n               =       v-m+1;

f               =       EM(tE,p(3),p(1),p(2))*dt;

A               =       zeros(v,m);
for i=1:m
    A(i:n+i-1,i)=       f;
end

error           =       (cOut-A*cIn);
end

%Engineering model
function E = EM(t,tau,n,tau_PFR)
tau_i             =       (tau-tau_PFR)/n;
E                 =       heaviside(t-tau_PFR).*(t-tau_PFR)...
                          .^(n-1)/(gamma(n)*tau_i^n).*...
                          exp(-(t-tau_PFR)/tau_i);
end
```

N.9 Data analysis rs-SDR

Listing N.38: Data analysis script for the RTD prediction from rs-SDR simulations

```
clear; close all; clc;
%% settings model

rd = 0.066; %disc radius [m]
rs = 0.067; %stator radius [m]
rshaft = 0.006; % shaft radius [m]
h = 2e-3;  % in 1 mm config: 1e-3 % gap height [m]
thickness_rotor = 8e-3; % Thickness rotor [m]
total_thickness = 0.012; % in 1 mm config: 0.01 [m]
G = 0.0303; % in 1 mm config: 1.52e-2
nu = 1e-6; % Kinematic viscosity water [m^2 s^-1]
omega = 50; % Rotational velocity [rad/s]
Q = 1e-5; % Injection flow rate [m^3/s]
Cw = 151; % Throughflow coeffiicent [-]
```

```

V_R = (pi*rs^2*total_thickness)-(pi*rd^2*thickness_rotor)-(pi*rshaft^2*h);

V_R_added = 5.12e-07; %% Added volume in tubing and injection zone
V_R = V_R + V_R_added;

t_RTD = linspace(0,25,1e4); % Time for simulation

% parameters for fitting
init_guess = [6 1.7 1];
LB = [0 0 0];
UB = [20 20 20];

%% 10 rad/s
% Results from experiments

V_PFR_per_V_R_10 = 0.215;
n_10 = 1.549;

% Reconstruct experimental RTD

tau_m = (V_R/Q);
tau_PFR_10 = V_PFR_per_V_R_10*tau_m;
tau_i_10 = (tau_m-tau_PFR_10)/n_10;

E_exp_10  =   heaviside(t_RTD-tau_PFR_10).*(t_RTD-tau_PFR_10)...
        .^(n_10-1)/(gamma(n_10)*tau_i_10^n_10).*...
        exp(-(t_RTD-tau_PFR_10)/tau_i_10);

% Analyzing the simulation data
Data_sim_10 = readmatrix(fullfile('RTD_10rads','surfaceFieldValue.dat'));

t_sim_10 = Data_sim_10(:,1);
Avg_outlet_10 = Data_sim_10(:,2);

RTD_sim_10 = gradient(Avg_outlet_10,t_sim_10);
RTD_sim_10 = movmean(RTD_sim_10,500);

% Fit the engineering model to the simulated RTD

param.Q = Q;
param.V_R = V_R;
param.RTD_sim = RTD_sim_10;
param.t_sim = Data_sim_10(:,1);

results_fit_10 = lsqnonlin(@fitcrit,init_guess,LB,UB,[],param);

tau_m_fit_10 = results_fit_10(1);

n_fit_10 = results_fit_10(2);
tau_PFR_fit_10 = results_fit_10(3);
V_PFR_norm_10 = tau_PFR_fit_10/tau_m_fit_10;

tau_i_fit_10  = (tau_m_fit_10-tau_PFR_fit_10)/n_fit_10;
E_fit_10      = heaviside(t_sim_10-tau_PFR_fit_10).*(t_sim_10-tau_PFR_fit_10)...
          .^(n_fit_10-1)/(gamma(n_fit_10)*tau_i_fit_10^n_fit_10).*...
          exp(-(t_sim_10-tau_PFR_fit_10)/tau_i_fit_10);

%% 25 rad/s
% Results from experiments

V_PFR_per_V_R_25 = 0.1733;
n_25 = 1.667233333;

% Reconstruct experimental RTD

tau_m = (V_R/Q);
tau_PFR_25 = V_PFR_per_V_R_25*tau_m;
tau_i_25 = (tau_m-tau_PFR_25)/n_25;

E_exp_25  = heaviside(t_RTD-tau_PFR_25).*(t_RTD-tau_PFR_25)...
        .^(n_25-1)/(gamma(n_25)*tau_i_25^n_25).*...
        exp(-(t_RTD-tau_PFR_25)/tau_i_25);
```

```

% Analyzing the simulation data
Data_sim_25 = readmatrix(fullfile('RTD_25rads','surfaceFieldValue.dat'));

t_sim_25 = Data_sim_25(:,1);
Avg_outlet_25 = Data_sim_25(:,2);

RTD_sim_25 = gradient(Avg_outlet_25,t_sim_25);
RTD_sim_25 = movmean(RTD_sim_25,500);

% Fit the engineering model to the simulated RTD

param.Q = Q;
param.V_R = V_R;
param.RTD_sim = RTD_sim_25;
param.t_sim = Data_sim_25(:,1);

results_fit_25 = lsqnonlin(@fitcrit,init_guess,LB,UB,[],param);

tau_m_fit_25 = results_fit_25(1);

n_fit_25 = results_fit_25(2);
tau_PFR_fit_25 = results_fit_25(3);
V_PFR_norm_25 = tau_PFR_fit_25/tau_m_fit_25;

tau_i_fit_25  = (tau_m_fit_25-tau_PFR_fit_25)/n_fit_25;
E_fit_25    = heaviside(t_sim_25-tau_PFR_fit_25).*(t_sim_25-tau_PFR_fit_25)...
          .^(n_fit_25-1)/(gamma(n_fit_25)*tau_i_fit_25^n_fit_25).*...
          exp(-(t_sim_25-tau_PFR_fit_25)/tau_i_fit_25);

%% 50 rad/s
% Results from experiments

V_PFR_per_V_R_50 = 0.1222;
n_50 = 1.778;

% Reconstruct experimental RTD

tau_m = (V_R/Q);
tau_PFR_50 = V_PFR_per_V_R_50*tau_m;
tau_i_50 = (tau_m-tau_PFR_50)/n_50;

E_exp_50  = heaviside(t_RTD-tau_PFR_50).*(t_RTD-tau_PFR_50)...
        .^(n_50-1)/(gamma(n_50)*tau_i_50^n_50).*...
        exp(-(t_RTD-tau_PFR_50)/tau_i_50);

% Analyzing the simulation data
Data_sim_50 = readmatrix(fullfile('RTD_50rads','surfaceFieldValue.dat'));

t_sim_50 = Data_sim_50(:,1);
Avg_outlet_50 = Data_sim_50(:,2);

RTD_sim_50 = gradient(Avg_outlet_50,t_sim_50);
RTD_sim_50 = movmean(RTD_sim_50,500);

% Fit the engineering model to the simulated RTD

param.Q = Q;
param.V_R = V_R;
param.RTD_sim = RTD_sim_50;
param.t_sim = Data_sim_50(:,1);

results_fit_50 = lsqnonlin(@fitcrit,init_guess,LB,UB,[],param);

tau_m_fit_50 = results_fit_50(1);

n_fit_50 = results_fit_50(2);
tau_PFR_fit_50 = results_fit_50(3);
V_PFR_norm_50 = tau_PFR_fit_50/tau_m_fit_50;

tau_i_fit_50  = (tau_m_fit_50-tau_PFR_fit_50)/n_fit_50;
E_fit_50    = heaviside(t_sim_50-tau_PFR_fit_50).*(t_sim_50-tau_PFR_fit_50)...
        .^(n_fit_50-1)/(gamma(n_fit_50)*tau_i_fit_50^n_fit_50).*...
        exp(-(t_sim_50-tau_PFR_fit_50)/tau_i_fit_50);

%% 75 rad/s
% Results from experiments

V_PFR_per_V_R_75 = 0.139033333;
n_75 = 1.5924;

% Reconstruct experimental RTD

tau_m = (V_R/Q);
tau_PFR_75 = V_PFR_per_V_R_75*tau_m;
tau_i_75 = (tau_m-tau_PFR_75)/n_75;

E_exp_75  = heaviside(t_RTD-tau_PFR_75).*(t_RTD-tau_PFR_75)...
        .^(n_75-1)/(gamma(n_75)*tau_i_75^n_75).*...
        exp(-(t_RTD-tau_PFR_75)/tau_i_75);

% Analyzing the simulation data
Data_sim_75 = readmatrix(fullfile('RTD_75rads','surfaceFieldValue.dat'));

t_sim_75 = Data_sim_75(:,1);
Avg_outlet_75 = Data_sim_75(:,2);

RTD_sim_75 = gradient(Avg_outlet_75,t_sim_75);
RTD_sim_75 = movmean(RTD_sim_75,500);

% Fit the engineering model to the simulated RTD

param.Q = Q;
param.V_R = V_R;
param.RTD_sim = RTD_sim_75;
param.t_sim = Data_sim_75(:,1);

results_fit_75 = lsqnonlin(@fitcrit,init_guess,LB,UB,[],param);

tau_m_fit_75 = results_fit_75(1);

n_fit_75 = results_fit_75(2);
tau_PFR_fit_75 = results_fit_75(3);
V_PFR_norm_75 = tau_PFR_fit_75/tau_m_fit_75;

tau_i_fit_75  = (tau_m_fit_75-tau_PFR_fit_75)/n_fit_75;
E_fit_75     = heaviside(t_sim_75-tau_PFR_fit_75).*(t_sim_75-tau_PFR_fit_75)...
          .^(n_fit_75-1)/(gamma(n_fit_75)*tau_i_fit_75^n_fit_75).*...
          exp(-(t_sim_75-tau_PFR_fit_75)/tau_i_fit_25);
%% compute integrals

Area_exp_10 = trapz(t_RTD,E_exp_10);
Area_sim_10 = trapz(t_sim_10,RTD_sim_10);
Area_fit_10 = trapz(t_sim_10,E_fit_10);

Area_exp_25 = trapz(t_RTD,E_exp_25);
Area_sim_25 = trapz(t_sim_25,RTD_sim_25);
Area_fit_25 = trapz(t_sim_25,E_fit_25);

Area_exp_50 = trapz(t_RTD,E_exp_50);
Area_sim_50 = trapz(t_sim_50,RTD_sim_50);
Area_fit_50 = trapz(t_sim_50,E_fit_50);

Area_exp_75 = trapz(t_RTD,E_exp_75);
Area_sim_75 = trapz(t_sim_75,RTD_sim_75);
Area_fit_75 = trapz(t_sim_75,E_fit_75);

%% renormalize
E_exp_10 = E_exp_10./Area_exp_10;
E_exp_25 = E_exp_25./Area_exp_25;
E_exp_50 = E_exp_50./Area_exp_50;
E_exp_75 = E_exp_75./Area_exp_75;

RTD_sim_10 = RTD_sim_10./Area_sim_10;
RTD_sim_25 = RTD_sim_25./Area_sim_25;
RTD_sim_50 = RTD_sim_50./Area_sim_50;
```

```matlab
RTD_sim_75 = RTD_sim_75./Area_sim_75;

E_fit_10 = E_fit_10./Area_fit_10;
E_fit_25 = E_fit_25./Area_fit_25;
E_fit_50 = E_fit_50./Area_fit_50;
E_fit_75 = E_fit_75./Area_fit_75;

%% plotting

figure(1)
plot(t_sim_25,Avg_outlet_25)
hold on
plot(t_sim_10,Avg_outlet_10)
plot(t_sim_50,Avg_outlet_50)
plot(t_sim_75,Avg_outlet_75)
hold off
ylabel('Outlet value [-]')
xlabel('Time [s]')
legend('25 rad/s','50 rad/s')

figure(2)
plot(t_RTD,E_exp_10,'-m')
hold on
plot(t_sim_10,RTD_sim_10,'--m')
plot(t_sim_10,E_fit_10,'-.m')
plot(t_RTD,E_exp_25,'-b')
plot(t_sim_25,RTD_sim_25,'--b')
plot(t_sim_25,E_fit_25,'-.b')
plot(t_RTD,E_exp_50,'-k')
plot(t_sim_50,RTD_sim_50,'--k')
plot(t_sim_50,E_fit_50,'-.k')
plot(t_RTD,E_exp_75,'-r')
plot(t_sim_75,RTD_sim_75,'--r')
plot(t_sim_50,E_fit_75,'-.-r')
hold off
ylabel('E(t)')
ylim([0 0.18])
xlabel('Time [s]')
legend('Experiment 10 rad/s','Experiment 25 rad/s','Simulation 25 rad/s','Fit 25
    rad/s','Experiment 50 rad/s','Simulation 50 rad/s','Fit 50 rad/s','Experiment
    75 rad/s', 'interpreter','latex')

%% functions

function [error] = fitcrit(x,param)

tau = x(1);
n = x(2);
tau_PFR = x(3);

%% Get simulation data
t_sim = param.t_sim;
RTD_sim = param.RTD_sim;

%% get the engineering model results
tau_i            =      (tau-tau_PFR)/n;
E                =      heaviside(t_sim-tau_PFR).*(t_sim-tau_PFR)...
                        .^(n-1)/(gamma(n)*tau_i^n).*...
                        exp(-(t_sim-tau_PFR)/tau_i);

error = abs(RTD_sim(:)-E(:));

end
```