University College of Southeast Norway

www.usn.no

FMH606 Master's Thesis

< CFD Validation of transient subsea gas plume model >

Taewook Kim

Faculty of Technology, Natural sciences and Maritime Sciences
Campus Porsgrunn

University College of Southeast Norway

www.usn.no

Course: FMH606 Master's Thesis

Title: CFD Validation of transient subsea gas plume model

Number of pages: 110pages

Keywords: <blowout,Computational Fluid dynmaics, CFD, Eulerian-lagrangian, ANSYS Fluent, stochastic tracking, randomwalk, integral model, OpenFoam, C++, C language, Matlab>

Student:	Taewook Kim
Supervisor:	Amaranath S.Kumara
External partner:	Lloyd's Register Consulting
Availability:	<Open/Confidential>

Approved for archiving: ______________________________
(supervisor signature)

Summary:

Subsea gas blowouts are often a significant risk contributor to offshore installations. It is important to understand plume that is created by the blowout. Different plume models were investigated. CFD model in ANSYS Fluent and integral model were performed to confirm its validity. CFD model in OpenFoam has also performed additionally.

For further validation, more experiments of larger depth and higher flow rate are necessary.

OpenFoam model needs to be modified.

Preface

This thesis is completed with the changed task description. Initially sea currents, change of density and size of bubbles and gas dissolution. However, it was decided not to include sea currents and gas dissolution in this thesis. Also, an analytical transient model named 'Rising cap model' that was provided by Lloyd's Register was supposed to be used. However, Rising cap model was not comparable to the CFD model that was used in this thesis for some reason. The reason is presented in the conclusion. Instead, I coded the steady state integral model. Additionally, I tried to develop an OpenFoam model, but judging from the results, the model needs to be developed further for validity.

The entire work was carried out from January 2017 to May 2017. The time is spent on literature reviews, CFD simulation, C programming, Matlab programming, C++ programming(OpenFoam).

I really appreciate Amaranath S. Kumara for opening this thesis topic and his great help and support throughout the period of this thesis.

Also thanks to Jan Erik Olsen who is senior research scientist at SINTEF for giving me great advice.

Lastly, I am grateful to Mikkel Bakli who is an advisor at Acona Flow Technology AS for helping me on CFD model. Without his help, I cannot imagine where this thesis would have ended up. Mange takk Mikkel.

Porsgrunn, 2017-05-14

Taewook Kim

Nomenclature

div	Divergence	-
$grad$	Gradient	-
$\vec{V}$	Velocity vector	m/s
t	Time	s
u	x-velocity	m/s
v	y-velocity	m/s
w	z-velocity	m/s
μ	Dynamic viscosity	$kg/m \cdot s$
S	Source	$kg \cdot m/s$
k_{therm}	Thermal conductivity	W/m^2k
i	Internal energy	J
φ	Variable	-
Γ	Diffusive term	-
$u(t)$	Velocity function of time	m/s
Φ	Dissipation function	$J \cdot m^3/s \cdot kg$
T	Temperature	k
U	Mean velocity	m/s
$u^{'}$	Fluctuating velocity	m/s
σ_k	Constant (1.00)	-
σ_ε	Constant (1.30)	-
$C_{1\varepsilon}$	Constant (1.44)	-

$C_{2\varepsilon}$	Constant (1.92)	-
S_{ij}	Mean rate of deformation	-
$\vec{F}$	additional acceleration	m/s^2
$\vec{F}_D$	Drag force	$1/s^2$
NP	Number of particle	-
$\vec{F}_v$	Force of virtual mass	m/s^2
$\dot{m}$	Mass rate	kg/s
m	mass	kg
C_{vm}	virtual mass factor(0.5)	-
ξ	normally distributed random number	-
T_L	Fluid Lagrangian integral time	s
ε	dissipation	m^2/s^3
k	Turbulent kinetic engergy	m^2/s^2
C_D	Drag coefficient	-
Re	Relative Reynolds number	-
E_0	Eötvös number	-
d	diameter	m
F_{other}	Other interaction forces	m/s^2
τ	relaxation time	s
C_1	coefficient	-
C_2	coefficient	μm

h	Height	m
R	Gas constant	$J / k \cdot mol$

Subscripts

c	centerline
a	atmospheric
~	Dimensionless value
g	gas
w	water
f	fountain
b	the bottom of the control volume
M	momentum
i	Internal energy
p	particle
t	turbulence
k	kinetic
ε	epsilon
fl	fluid
0	Initial
hd	Hydrostatic
r,q	r-th phase, q-th phase
br	For breakup
co	For coalescence
eq	Equilibrium

Contents

1 Introduction

The early studies of underwater plumes were motivated by the interest in uncontrolled blowouts resulting from accidents in offshore drilling or broken gas pipelines. Potentially it could be a danger to ships and offshore structures. The main reason of the sinking of floating structure above subsea blowouts is considered mainly to be caused by radial water currents at the sea surface. As another consequence of blowouts, ignition of the flammable vapor leakage can cause structural damage to the platform. The purpose of modeling the subsea dispersion is to provide properties for input data for models that quantify the above hazard.

To predict plume dynamics, mainly two different modeling method have been used.

Integral models have been developed by many authors mostly based on Taylor's idea[1] as a derived model, Friedel's model is introduced in this thesis.

As an alternative way to predict plume, blowout can be modeled using CFD. As one of the CFD models, Cloete introduced CFD model[2] that was validated comparing with Engebretsen's experiment[3]. The simulation of the model is introduced in this thesis, validating the model comparing with experimental data of different conditions.

Integral model and CFD model are compared for validity on the same experiments.

OpenFoam simulation is attempted to make the same model of Cloete's.

2 Plume modeling

2.1 Type of models

The framework illustrated in Figure 2.1 is typically used. The dispersion of the gas from the release point to the surface is considered in three zones:

Zone of Flow Establishment(ZOFE) is the region between the release point and the height where the dispersion appears to build a plume structure. At this height, the effect of buoyancy is more prevailing than initial release momentum. Zone of Established Flow(ZOEF) is the plume-like region that is extended from the ZOFE up to a depth which is beneath the free surface by approximately one plume diameter. Zone of Surface Flow(ZOSF) is the region above the ZOEF where the plume interacts with the surface where the bubble plume and radial flow of water at the surface widen.

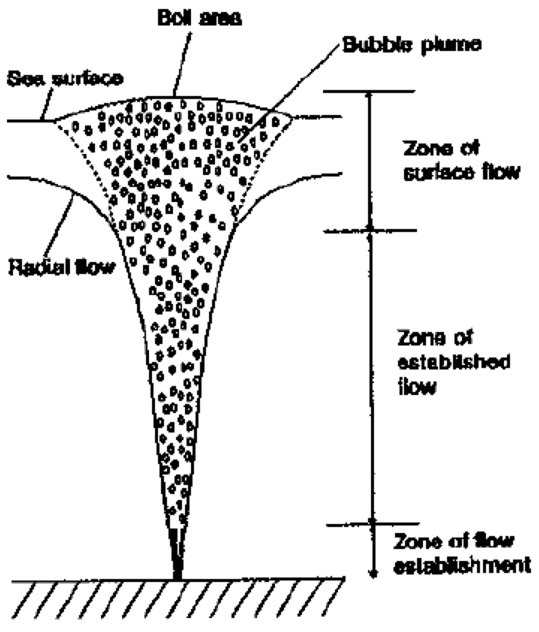

Figure 2.1: Typical plume model representing zone of flow establishment, zone of established flow and zone of surface flow

Three approaches of different complexity have been used in modeling the dispersion of subsea release. The empirical model is the simplest one that is assumed that the plume radius is proportional to the release depth or correlations. Another approach is an integral type model which is based on local similarity. For instance, a velocity profile is assumed to have a similar form at different heights. The plume properties can be well described by Gaussian profiles. Entrainment of water into the plume is described by the use of an entrainment coefficient. Specific of integral models are introduced in chapter 2.2.

The most complex models are represented by Computational Fluid Dynamics(CFD) or field codes by solving Navier Stokes Equations. Their advantage over integral models is that CFD models do not require the use of empirical constants. CFD model is introduced from chapter 3 and 4.

2.2 Integral model (M.J. Friedl -2000)

M.J. Friedl presented an integral model in 2000[4]. A sketch of the model is described in Figure 2.2. The main purpose of the model was to develop a theoretical model for the fountain and to overcome the problem associated with scaling from small scale to full scale. The assumptions and simplifications of the model are discussed briefly in the following. The details are described in M.J. Friedl's work.

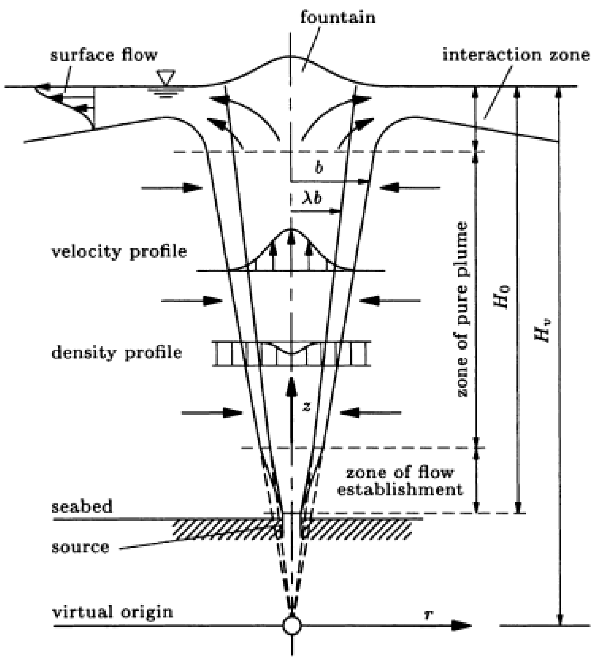

Figure 2.2: Sketch of the integral model

2.2.1 Assumptions

The mean flow is stationary. The stagnant water is assumed. Incompressible continuous phase is assumed.

Constant average bubble diameter of 1cm is assumed. In the model, bubble size is not as important as the buoyancy and its radial distribution. Constant diameter is linked to constant slip velocity[1] of 0.35m/s.

The profile of velocity and the void fraction is assumed to have Gaussian shapes. Entrainment coefficient (α) is constant. Entrainment coefficient is the proportionality of the rate of entrainment to the local velocity. The proportionality between the momentum carried by the mean vertical velocity and the fluctuating vertical velocity is constant (γ).

[1] The difference between the average velocities of two different fluids flowing together

2.2.2 Theory of bubble plumes

The governing equations are expressed in cylindrical coordinates.

The vertical velocity and void fraction are shown below in Equation (2.1) and (2.2) respectively.

$$v(r,z) = v_c(z)e^{-(r^2/b(z)^2)} \tag{2.1}$$

$$\phi(r,z) = \phi_c(z)e^{-r^2/(\lambda b(z))^2} \tag{2.2}$$

Where λ is the ratio of the widths as shown in Figure 2.2. Subscript c represents centerline value.

The length scale H_p is introduced in the model. This term represents the water depth corresponding to the atmospheric pressure P_0. The relation is such

$$H_p = \frac{P_a}{\rho g} \tag{2.3}$$

In the case of fresh water, the value is 10.33m, for the case of ocean water, the value is slightly below 10 m at the standard atmospheric condition.

The dimensionless axial coordinate $\tilde{z}$, the dimensionless width of the plume $\tilde{b}$, and the dimensionless vertical liquid velocity $\tilde{v}$ are defined.

$$\tilde{z} = \frac{z}{H_v + H_p} \tag{2.4}$$

$$\tilde{b} = \frac{b}{2\alpha(H_v + H_p)} \tag{2.5}$$

$$\tilde{v} = v\left(\frac{1+\lambda^2}{2\pi\gamma\alpha^2}\frac{g\dot{V}_g}{H_v + H_p}\right)^{-1/3} \tag{2.6}$$

g is gravity, $\dot{V}_g$ is volume flow rate of gas and H_v is water depth.

The continuity equation for the gas phase is

$$\tilde{\phi}_c(\tilde{z}) = \frac{1}{1-\tilde{z}}\frac{1}{\tilde{b}^2(\tilde{v}_c + \tilde{s})} \tag{2.7}$$

$$\tilde{\phi}_c(\tilde{z}) = \phi_c\left(\frac{(1+\lambda^2)^2\gamma}{\pi^2\lambda^2 2^5\alpha^4}\frac{\dot{V}_g^2}{(H_v + H_p)^5 g}\right)^{-1/3} \tag{2.8}$$

where γ is momentum amplification factor. And $\tilde{s}$ is the influence of slip velocity in dimensionless form, which is

$$\tilde{s} = (1+\lambda^2)\tilde{v}_s \tag{2.9}$$

Continuity eqaution of the liquid phase and the momentum equation for the mixture is

$$\frac{d}{d\tilde{z}}(\tilde{b}^2\tilde{v}_c) = \tilde{b}\tilde{v}_c \tag{2.10}$$

$$\frac{d}{d\tilde{z}}(\tilde{b}^2\tilde{v}_c^{\,2}) = \frac{1}{(\tilde{v}_c+\tilde{s})(1-\tilde{z})} \tag{2.11}$$

In the model, by solving these two equations by using a newly introduced approximate procedure, it allows to predict large scale plumes from the data of laboratory scale.

And the approximate solutions for the width and velocity are derived using a perturbation technique as shown in Equation (2.12) and (2.13)

$$\begin{aligned}\tilde{b}(\tilde{z}) = {} & \frac{3}{5}\tilde{z}\left[1-\frac{\tilde{z}}{13}-7\left(\frac{\tilde{z}}{13}\right)^2+\cdots\right]+\tilde{s}\,\frac{3}{110}\left(\frac{12}{25}\right)^{1/3}\tilde{z}^{4/3}\left[1-\frac{1046}{49}\frac{\tilde{z}}{39}\right.\\ & \left.-\frac{227726}{833}\left(\frac{\tilde{z}}{39}\right)^2+\cdots\right]-\tilde{s}^2\,\frac{48}{15125}\left(\frac{25}{12}\right)^{1/3}\tilde{z}^{5/3}\left[1-\frac{34663}{9408}\tilde{z}+\frac{225707803}{240143904}\tilde{z}^2+\cdots\right]+\cdots\end{aligned} \tag{2.12}$$

$$\begin{aligned}\tilde{v}_c(\tilde{z}) = {} & \left(\frac{25}{12}\right)^{1/3}\tilde{z}^{-1/3}\left[1+11\frac{\tilde{z}}{39}+\frac{511}{2}\left(\frac{\tilde{z}}{39}\right)^2+\cdots\right]-\tilde{s}\,\frac{7}{22}\left[1+\frac{345}{343}\frac{\tilde{z}}{13}+\frac{86175}{11662}\left(\frac{\tilde{z}}{13}\right)^2\right.\\ & \left.+\cdots\right]+\tilde{s}^2\,\frac{13}{121}\left(\frac{12}{25}\right)^{1/3}\tilde{z}^{1/2}\left[1-\frac{59489}{1456}\frac{\tilde{z}}{39}-\frac{2835583625}{23347324}\left(\frac{\tilde{z}}{39}\right)^2+\cdots\right]\cdot+\cdots\end{aligned} \tag{2.13}$$

2.2.3 Theory of fountains

The fountain can be modeled by using momentum balance. The momentums acting on a plume is described in Figure 2.3. In the model, it was assumed that the control volume is infinite laterally, so no vertical momentum flux pass through the outer boundaries. The forces apply to the control volume are atmospheric pressure(p_a) integrated over the water surface, pressure at the bottom of the control volume(p_b) integrated over the bottom boundary, weight of the water masses(G_w) below the level of the quiescent surface, weight of the fountain(G_f) and total buoyancy(B). The balance equation is such

$$p_a + G_w + G_f = p_b + B \tag{2.14}$$

If the lower boundary of the control volume is set below the release point of the plume, the p_b will not be affected by the plume. Meaning that it can be modeled by neglecting terms that cancel out each other which are p_a, p_b and G_w. Then Equation (2.14) yields

$$G_f = B \tag{2.15}$$

By defining G_f with density(ρ_f) and volume(V_f), it leads to

$$g\rho_f V_f = B \tag{2.16}$$

By defining B is equal to the theoretical momentum flux $\rho(z)\pi\gamma(1/2)b^2(z){v_c}^2(z)$ at $z = H_v$ and $V_f = \pi b_f^2 h_f$, Equation (2.16) yields

$$g\rho_f \pi b_f^2 h_f = \rho(H_v)\pi\gamma\frac{1}{2}b^2(H_v)v_c^2(H_v) \tag{2.17}$$

The density of the fountain ρ_f and the density of the plume at the level of the quiescent surface $\rho(H_v)$ are assumed to be equal. Also, the kinetic energy at the base was assumed to be converted to potential energy. So, Equation (2.17) is simplified as

$$\beta\gamma v^2 = gh_f \tag{2.18}$$

Where $v = v_c(H_v)$ is the velocity of the fluid particle at the fountain's base and h_f is its height. As a function of r, it is given by

$$h(r) = h_f e^{-r^2/b_f^2} - h_{offset} \tag{2.19}$$

Where h_{offset} is the offset of the Gaussian profile baseline with respect to the level of the quiescent water surface.

This model is coded in Matlab as attached in Appendix B-1. The applications of this codes are shown in chapter 4 and 5.

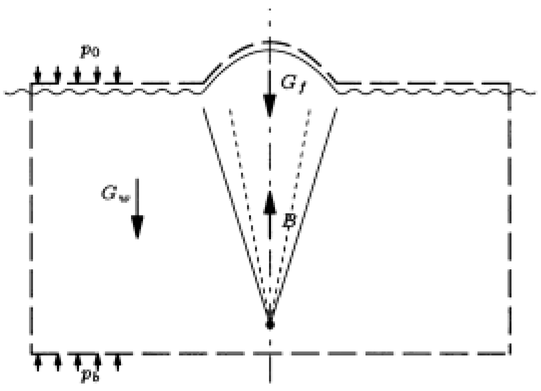

Figure 2.3: The momentums acting on a plume

3 Basic theories of CFD

Computational fluid dynamics(CFD) is the way of calculating fluid flow, heat transfer etc. by solving Navier Stokes equations in discretized domain. The basic theory of Navier Stokes equations and discretized method will be briefly introduced in the following sub-chapters. Also, theoretical background of models that are used in validation cases in further chapters.

3.1 Discretization

The main idea of discretization is to convert continuous domain to a discrete domain using a grid. By having discretized domain, it enables to obtain approximated algebraic equations from Navier Stokes equations. In other words, this conversion simply breaks down Navier Stokes equations that are non-solvable partial differential equation into solvable equations. There are three different discretization methods being used representatively, such as finite difference method, finite element method and finite volume method. In this thesis, Finite Volume Method(FVM) is the main interest since CFD programs that are used in this thesis are based on finite volume method.

In FVM, the domain is divided into cells where the variables are stored at the center of the cell. Then the governing equations are solved over each cell. Interpolations will be carried out in order to describe variables between cell centers. The example of FVM is shown in Figure 3.1.

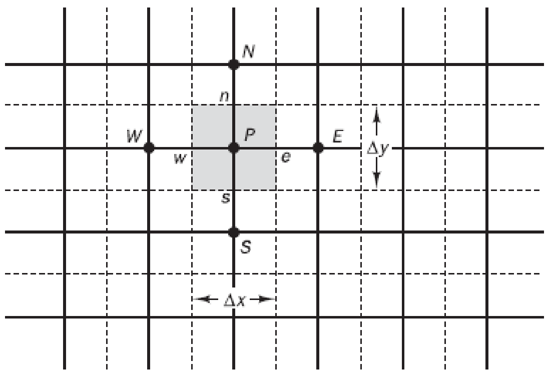

Figure 3.1: Example of finite volume method

3.2 Navier Stokes equations

The Navier-Stokes equations govern the motion of a viscous and heat conducting fluid. Strictly speaking, Navier Stokes equations describe the momentum of the fluid, however, in the modern CFD literature, this terminology is regarded as it includes continuity equation and energy equation as well. The governing equations represent the conservation laws of physics: The mass of a fluid is conserved (Continuity equation). The change rate of momentum is equal to the sum of the forces on a fluid particle (Momentum equation). The change rate of energy equals the sum of the rate of heat/work addition to a fluid particle (Energy equation). Continuity equation, Momentum equation, and Energy equation that are in the form for finite volume method are shown below in Equation (3.1), (3.2) and (3.3) respectively.

$$\frac{\partial \rho}{\partial t} + div(\rho \vec{V}) = 0 \tag{3.1}$$

$$\frac{\partial (\rho u)}{\partial t} + div(\rho u \vec{V}) = -\frac{\partial p}{\partial x} + div(\mu \; grad \; u) + S_{Mx}$$

$$\frac{\partial (\rho v)}{\partial t} + div(\rho v \vec{V}) = -\frac{\partial p}{\partial y} + div(\mu \; grad \; v) + S_{My} \tag{3.2}$$

$$\frac{\partial (\rho w)}{\partial t} + div(\rho w \vec{V}) = -\frac{\partial p}{\partial z} + div(\mu \; grad \; w) + S_{Mz}$$

$$\frac{\partial (\rho i)}{\partial t} + div(\rho i \vec{V}) = -p \; div \vec{V} + div(k_{therm} \; grad \; T) + \Phi + S_i \tag{3.3}$$

Navier stokes equations are closely related to the transport equation. In the transport equation, the general variable φ is introduced, including equations for scalar quantities such as temperature and pollutant concentration etc., is given in Equation (3.4). The first term on the left-hand side accounts for transient accumulation, the second one for convection which is a transport of φ due to velocity. Γ is the diffusive term. On the right-hand side, the first term accounts for diffusion due to gradient and the last term is source of φ.

$$\frac{\partial (\rho \varphi)}{\partial t} + div(\rho \varphi \vec{V}) = div(\Gamma \; grad \; \varphi) + S_{\varphi} \tag{3.4}$$

3.3 Turbulence model

At low Reynolds numbers, flows are laminar which is stable flow type that occurs when a fluid flow is in parallel layers, with no disruption between the layers. In the meantime, at high Reynolds numbers, flows become turbulent which is a chaotic and random state of motion. In the chaotic state, velocity and pressure change continuously over time. An example of both flows is seen in Figure 3.2.

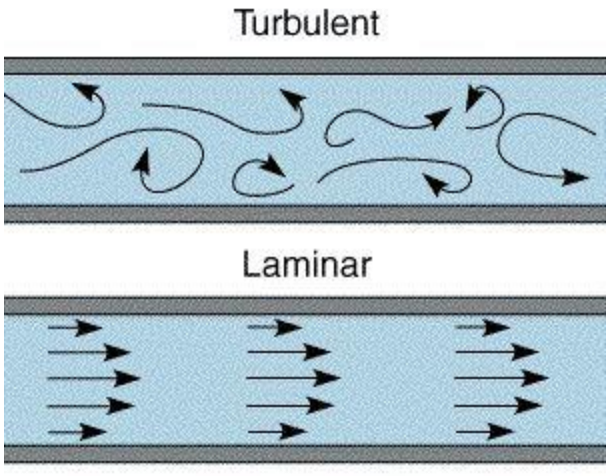

Figure 3.2: Example of Turbulent flow and laminar flow

As a physical description, turbulent flows are irregular. This makes turbulence problems impossible to solve deterministically. Therefore, statistical approaches and stochastic processes are required for solving turbulence problems such as ensemble average that is introduced in the later paragraph and stochastic tracking (random walk model) that is used for tracking turbulent particles in the next chapter.

Another physical description of turbulent flow is diffusivity. The diffusivity of turbulence increases rates of momentum, mass and heat transfer and make good mixing of the fluid.

Due to these reasons that account for the huge influence of turbulence, it is crucially important to capture turbulence. There are three representative methods for capturing/modeling turbulence.

Before introducing the methods, it is necessary to understand turbulence with a statistical approach. As described above, turbulent flows are chaotic and random. This randomness of the flow takes place in flow variables such as velocity as shown in the figure below.

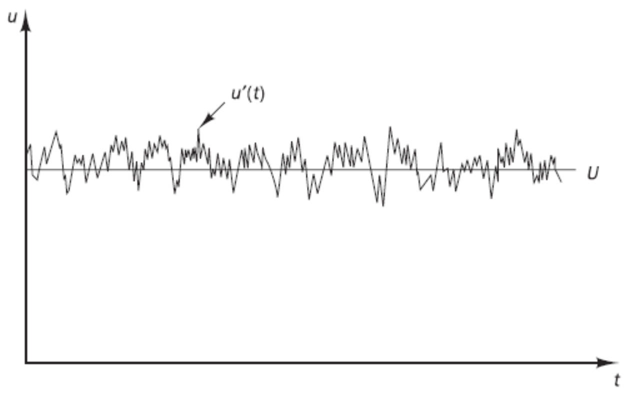

Figure 3.3: Fluctuating velocity of a turbulent flow over time

To simplify this randomness, a statistical way is widely used by decomposing flow variables into the mean value (U) and the fluctuating value ($u^{'}(t)$) as shown in the equation below. The mean value here is called ensemble average[2]. And this decomposition process is called Reynolds decomposition.

$$u(t) = U + u^{'}(t) \tag{3.5}$$

The first method of modeling turbulence is to use Reynolds-averaged Navier–Stokes(RANS) equation. In RANS, instead of including all the fluctuation into the calculation, it takes only mean value U with additional transport equations.

Another method is Large eddy simulation(LES). In this method, small eddies are neglected, only large eddies are tracked.

Lastly, there is Direct Numerical Simulation(DNS). In DNS, the mean velocity and all turbulent velocity fluctuations are computed. Since the simulation computes small eddies as well, it needs very fine grids for the small eddies. So that it can solve the Kolmogorov length scales where energy dissipation occurs and with a small time step to solve the fastest fluctuations.

LES and DNS are relatively costly compared to RANS. RANS is reasonably accurate, the computational cost is reasonable. Therefore, RANS(k-epsilon) is used in this thesis.

3.3.1 K-epsilon model

RANS turbulence models have additional transport equations accounting for turbulence, except Mixing length model has zero extra transport equation. K-epsilon model has 2 additional equations

The standard k-epsilon model is a semi-empirical model based on transport equation for turbulent kinetic energy and its dissipation rate epsilon. The two equations are shown below.

$$\frac{\partial(\rho k)}{\partial t} + div(\rho k\vec{u}) = div(\frac{\mu_t}{\sigma_k}\, grad\, k) + 2\mu_t S_{ij} \cdot S_{ij} - \rho\varepsilon \tag{3.6}$$

$$\frac{\partial(\rho \varepsilon)}{\partial t} + div(\rho \varepsilon\vec{u}) = div(\frac{\mu_t}{\sigma_\varepsilon}\, grad\, \varepsilon) + C_{1\varepsilon}\frac{\varepsilon}{k} 2\mu_t S_{ij} \cdot S_{ij} - C_{2\varepsilon}\rho\frac{\varepsilon^2}{k} \tag{3.7}$$

Where σ_k =1.00, σ_ε =1.30, $C_{1\varepsilon}$ =1.44 and $C_{2\varepsilon}$ =1.92. In the derivation of the k $-\varepsilon$ model, it was assumed that the flow is fully turbulent, and the effects of molecular viscosity are negligible. The standard k$-\varepsilon$ model is, therefore, valid only for fully turbulent flows. The drawback of this model is delayed and reduced separation.

[2] Esemble average $X \equiv \lim_{N\to\infty}\frac{1}{N}\sum_{n=1}^{N} x_n$, where N is the number of case/trial and x_n is a variable at n-th case/trial

3.4 Euler-Lagrange approach

In fluid mechanics, Lagrangian description or the Eulerian description is used to define a flow. In Lagrangian description, a fluid consists of particles which carry its own properties (velocity, pressure etc.). The way of describing the flow is to track the detailed histories of each fluid particle. The properties of particles are a function of time such as $\rho(t)$, $\vec{u}(t)$, $p(t)$, etc. Simply put, conservation of mass and Newton's laws are employed to each particle.

In Eulerian description, instead of tracking each particle, the fluid properties are recorded as the function of location and time such as $\rho(\vec{x},t)$, $\vec{u}(\vec{x},t)$, $p(\vec{x},t)$, etc.

Lagrangian description is computationally expensive since each particle needs to be tracked while Eulerian description is mostly used in fluid mechanics. An example of comparison between Eulerian description and Lagrangian description are shown as the temperature of smoke coming out a chimney in Figure 3.4.

In the CFD model, Eulerian and Lagrangian description are coupled, meaning that Eulerian description is implemented for the fluid phase (sea and atmosphere for the case) as it is treated as a continuum by solving the Navier Stokes equations. On the other hand, Lagrangian description applies to the dispersed phase (gas bubbles for the case) as it is solved by tracking the dispersed phase. The dispersed phase and the fluid phase interact each other by exchanging momentum, mass, and energy.

Using the coupled method is computationally cheap and more accurate than using Euler-Euler approach, even though it is mentioned above that using Lagrangian description is computationally expensive. The reason is that, in the case where bubbles are injected, the grid size is decided related to the gas bubble size in Euler-Euler approach. It needs very fine mesh cells compared to Euler-Lagrange approach.

In ANSYS Fluent, the coupled approach was simply achieved by selecting 'Discrete Phase Model' and 'Volume of Fluid'. In the sub-chapters below, theories of the functions in ANSYS Fluent, that also were used for the CFD model in the thesis will be explained.

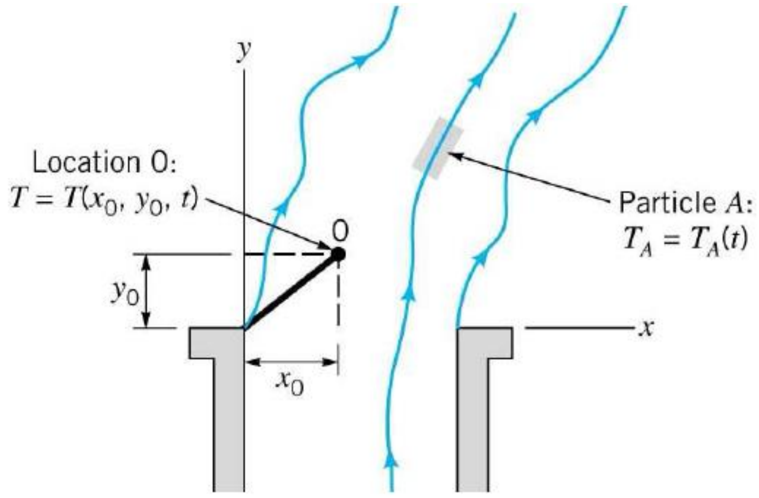

Figure 3.4: Example of Eulerian description and Lagrangian description

3.4.1 Discrete phase model(DPM)

DPM tracks particles by applying the particle force balance as described in Equation (3.8).

$$\frac{d\vec{u}_p}{dt} = F_D(\vec{u}_{fl} - \vec{u}_p) + \frac{\vec{g}(\rho_p - \rho_{fl})}{\rho_p} + \vec{F} \tag{3.8}$$

This force balance accounts for the particle inertia with the forces acting on the particle[5]. On the right side, the first term is the drag force per unit particle mass, the second term is a gravity term and the third term is an additional acceleration (buoyancy, lift, etc.).

3.4.1.1 Parcels

The mass flow rate of particle injection needs to be introduced, and this mass flow rate will determine the number of particles. As a concept of Lagrangian description, the DPM model tracks particles. Instead of tracking each particle, the model tracks 'parcel'. The parcel is representative of a fraction of the total continuous mass flow rate in steady tracking or a fraction of the total mass flow released in a time step in unsteady tracking. The reason why parcel is representative concept is because particles in a parcel share the same properties.

By decreasing the number of objects to track by implementing the concept of 'parcel', it makes DPM simulation computationally affordable. For instance, for the CFD validation in the next chapter, the case where mass flow rate is 0.208kg/s with 10 injections streams, timestep is 0.01s, and mass per particle is 8.03906E-08kg, the number of particles in a parcel is

$$NP = \dot{m}_{stream} \frac{\Delta t}{m_p} = \frac{0.208kg/s}{10} \cdot \frac{0.01s}{8.03906e-08kg} \approx 2587 \tag{3.9}$$

In short, without using parcels, the computational cost would 2587 times more expensive.

3.4.1.2 Virtual mass

For the unsteady motion of bodies underwater or unsteady flow around objects, additional effect(force) resulting from the fluid, acting on the structure needs to be considered. This added effect is called virtual mass or added mass. This added mass is the weight added by an accelerating or decelerating body affects surrounding fluid.

In Fluent, the force of virtual mass is written as the equation below

$$\vec{F}_v = C_{vm} \frac{\rho}{\rho_p} \left(\vec{u}_p grad\, \vec{u} - \frac{d\vec{u}_p}{dt} \right) \tag{3.10}$$

C_{vm} is the virtual mass factor with a default value of 0.5

3.4.1.3 Random walk model

As explained in chapter 3.3, Reynolds Averaged Navier Stokes equations can not provide the instantaneous velocity. Therefore the fluctuating velocity has to be estimated by stochastic tracking to model turbulent dispersion. In Fluent, Discrete Random Walk model(DRW) is used for stochastic tracking.

DRW is a sort of Random Walk model that is rather mathematical or statistical concept. In Random Walk model, it assumes situations where an object moves in a sequence of steps in randomly chosen directions. Many phenomena can be modeled as a random walk. In the figure below, it shows a simple example of random walks where going to the right and the left have the same possibility.

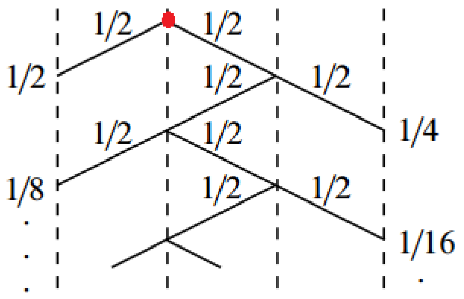

Figure 3.5: Example of random walk

In DRW, or Eddy Intergration Model(EIM), each eddy[3] is characterized by a Gaussian distributed random velocity fluctuation, u', v', w' in cartesian coordinates and a time scale τ_e. The velocity fluctuation components occur during the lifetime of the turbulent eddy. Each component is sampled with an assumption that they will obey a Gaussian probability distribution. They are given by

$$u' = \xi\sqrt{\overline{u'^2}}, \quad v' = \xi\sqrt{\overline{v'^2}}, \quad w' = \xi\sqrt{\overline{w'^2}} \tag{3.11}$$

[3] eddy is the swirling of a fluid and the reverse current created when the fluid is in a turbulent flow regime

Where ξ is normally distributed random number. The remainder of the right-hand side is the local RMS value of the velocity fluctuations. It is assumed that the fluctuating values are isotropic, meaning that it has no preferred direction, they can be defined for k-epsilon, k-omega and their variants as

$$\sqrt{\overline{u'^2}} = \sqrt{\overline{v'^2}} = \sqrt{\overline{w'^2}} = \sqrt{2k/3} \tag{3.12}$$

These fluctuating compnents are kept constant over an interval of time given by the characteristic lifetime of the eddies. The characteristic lifetime of the eddy is defined as

$$\tau_e = 2T_L \tag{3.13}$$

Where T_L is the fluid Lagrangian integral time. For the k-epsilon models and its variants

$$T_L = 0.15\frac{k}{\varepsilon} \tag{3.14}$$

In Fluent, the trajectory equation(Equation (3.8)) for individual particles will be integrated along the particle path to include the random effects of turbulence that are described above.

3.4.1.4 Drag laws

In particle tracking in the CFD model that is introduced in chapter 4, the lift force was neglected due to its minor contribution to the force balance [2]. Gravity and buoyancy are automatically calculated in ANSYS Fluent. In the meantime, the drag force should be included either by basic drag laws that Fluent provides, or user-defined function.

Several laws for drag coefficients applied to the CFD validation in chapter 4.

3.4.1.4.1 Spherical drag law

This drag law applies to particles of spherical shape. The relationship between drag coefficient (C_D) and Relative Reynolds number(Re) are approximated to the standard drag curve. The curve is described in Figure 3.6.

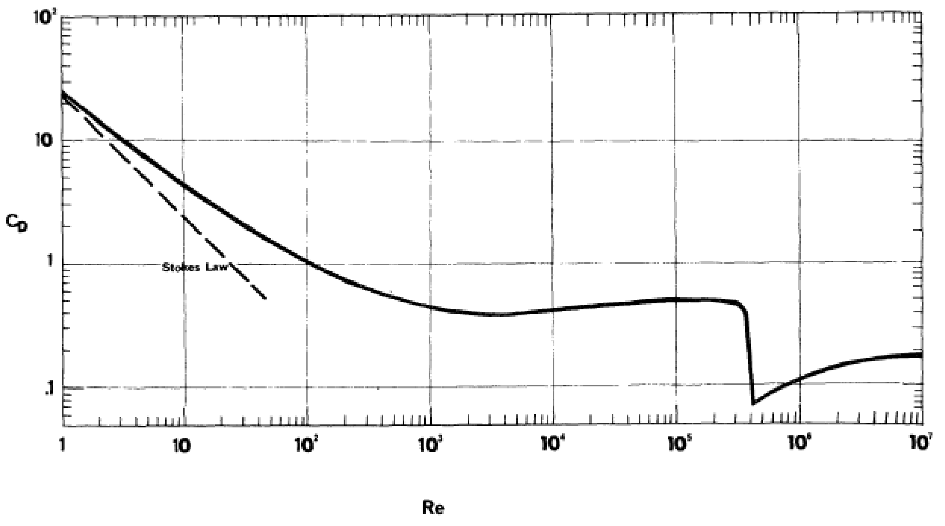

Figure 3.6: Drag coefficient for spherical particles vs. Reynolds number

C_D for smooth particles can be taken as shown below

$$C_D = a_1 + \frac{a_2}{\mathrm{Re}} + \frac{a_3}{\mathrm{Re}^3} \tag{3.15}$$

a_1, a_2 and a_3 are given by Morsi and Alexander as shown in Appendix A.

This drag law can be easily selected as in-built function.

3.4.1.4.2 Modified Spherical Drag law

This drag law is also based on the drag on the standard drag curve developed by J.R. Grace and M.E. Weber[6]. Many empirical data have been proposed to approximate this curve. The difference between this drag and the spherical drag is that this drag law is more specified over high Reynolds number. Correlations are shown in Appendix B. The law is coded as a UDF in Fluent. The code is shown in Appendix C.

3.4.1.4.3 Xia's drag law

The shape of bubble has significant influence on drag force acting on rising bubble. Xia[7]'s simulation comparing with the experimental data using a liquid metal. And bubbles in a liquid metal rather have large sizes and distorted shape in the turbulent region. Therefore, the drag law that is validated in Xia's work can account for larger bubbles that are deformed from the spherical shape.

The main relations are shown in the equations below, and originally proposed by Hamathy [8].

$$C_D = \frac{2}{3}\sqrt{\frac{E_0}{3}}, \quad E_0 = \frac{g(\rho_q - \rho_p)d_p^2}{\sigma} \tag{3.16}$$

Where E_0 is Eötvös number. It is a dimensionless number that characterizes the shape of bubbles. This drag law was used by Cloete's validation[2]. The UDF was provided by Mikkel[9].

3.4.1.4.4 Tomiyama's drag law

Tomiyama's model is well suited to gas-liquid flow where the bubbles can have a range of shapes[5]. The main relations are shown in the equations below.

$$C_D = \max\left(\min\left(\frac{24}{\text{Re}}\left(1+0.15\text{Re}^{0.687}\right), \frac{72}{\text{Re}} \right), \frac{8}{3}\frac{E_0}{E_0+4} \right) \tag{3.17}$$

The UDF of Tomiyama's drag is shown in Appendix B.

3.4.1.5 Two-way coupling

In the coupled approach or two-way coupling, continuous phase flow pattern and the discrete phase impact each other. This coupling is achieved by solving the discrete and continuous phase equations until the solutions in both phases stop changing. It is described in Figure 3.7.

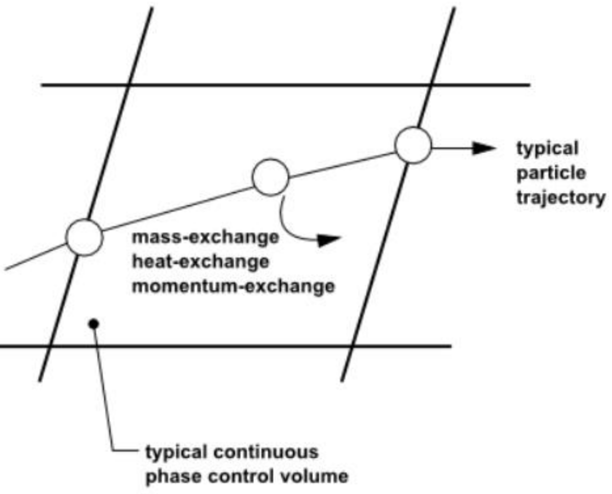
typical particle trajectory
mass-exchange
heat-exchange
momentum-exchange
typical continuous phase control volume

Figure 3.7: Interaction between particle and continuous phase

The transfer of momentum, mass and heat from the continuous phase to the discrete phase is computed in ANSYS Fluent. However, the heat transfer was not the main interested in the CFD model in the next chapters. The explanation of heat will be neglected in this thesis. The momentum change and the mass change are computed as described below

$$F = \sum\left(\frac{18\mu C_D \text{ Re}}{\rho_p {d_p}^2 24}\left(u_p - u_{fl}\right) + F_{other} \right)\dot{m}_p \,\Delta t \tag{3.18}$$

$$M = \frac{\Delta m_p}{m_{p,0}}\dot{m}_{p,0} \tag{3.19}$$

3.4.2 Volume of fluid(VOF)

VOF model can model multiple immiscible fluids by solving momentum equations and tracking the volume of fraction. If the qth fluid's volume fraction in the cell is denoted as ϕ_q

then when $0\langle\phi_q\langle 1$, the cell has the interface between qth fluid and one or more fluids.(when $\phi_q = 0$, the cell is empty of qth fluid, and when $\phi_q = 1$, the cell is full of qth fluid.

The tracking of the interface(s) between the phases is accomplished by solving a continuity equation for the volume fraction of each phase. For the qth fluid, volume fraction equation is

$$\frac{1}{\rho_q}\left[\frac{\partial(\phi_q\rho_q)}{\partial t} + div(\phi_q\rho_q\overrightarrow{V_q}) = S_{\phi_q} + \sum_{p=1}^{n}(\dot{m}_{rq} - \dot{m}_{qr})\right] \tag{3.20}$$

Where $\dot{m}_{pq}$ is the mass transfer from phase q to phase r and $\dot{m}_{qr}$ is the mass transfer in the otherwise direction. S_{ϕ_q} is source term.

The volume fraction equation will be solved for the primary phase. It will be solved for secondary, tertiary,..., n-th phase while the primary phase is constrained by the equation below

$$\sum_{q=1}^{n}\phi_q = 1 \tag{3.21}$$

As for momentum, a single set of momentum equation is solved, and the resulting velocity will be shared among the phases via ρ and μ.

3.5 Scheme

To calculate gradients and fluxes at the control volume faces, an approximate distribution of properties between nodal points is required. An example of 1-D grid is given in Figure3.8, discretized by the finite volume method. If steady convection-diffusion condition is assumed, the transport equation yields

$$div(\rho u\varphi) = div(\Gamma\ grad\ \varphi) + S_\varphi \tag{3.22}$$

Nodal points (W, P, E) that are placed in the center of the control volume store variables of interest. Through the calculation by using the transport equation, it is necessary to integrate the equation over the control volume. Then diffusive coefficient Γ and variable φ at the cell faces (w, e) are required.

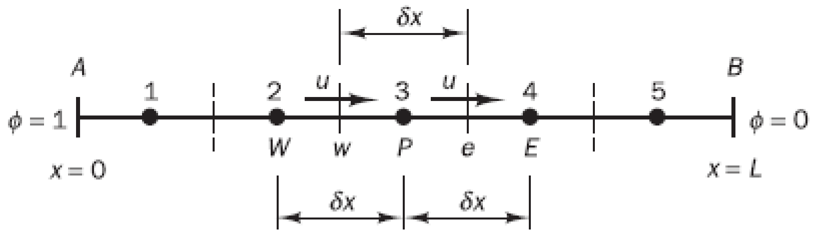

Figure 3.8: Example of 1-D grid

As the simplest method of approximating values at the cell faces is the central differencing scheme. For instance, the middle value between W and P can be taken at the cell face w.

Another representative scheme is the first-order upwind differencing scheme. In the scheme, it is assumed that the flow direction determines the value at a cell face. For instance, the value at W can be taken for the value at w.

The drawback of the first-order upwind scheme is inaccuracy. For more accuracy, the second-order upwind differencing scheme can be used.

An example of comparison between those schemes presented above is shown in Figure 3.9. The example is the transportaion of φ by convection-diffusion through the one-dimensional domain sketched in Figure 3.8 with $u = 2.5m/s$, $L = 1.0m$, $\rho = 1.0kg/m^3$ and $\Gamma = 0.1kg/m \cdot s$

Matlab code for three schemes are attached in Appendix B-2, B-3 and B-4.

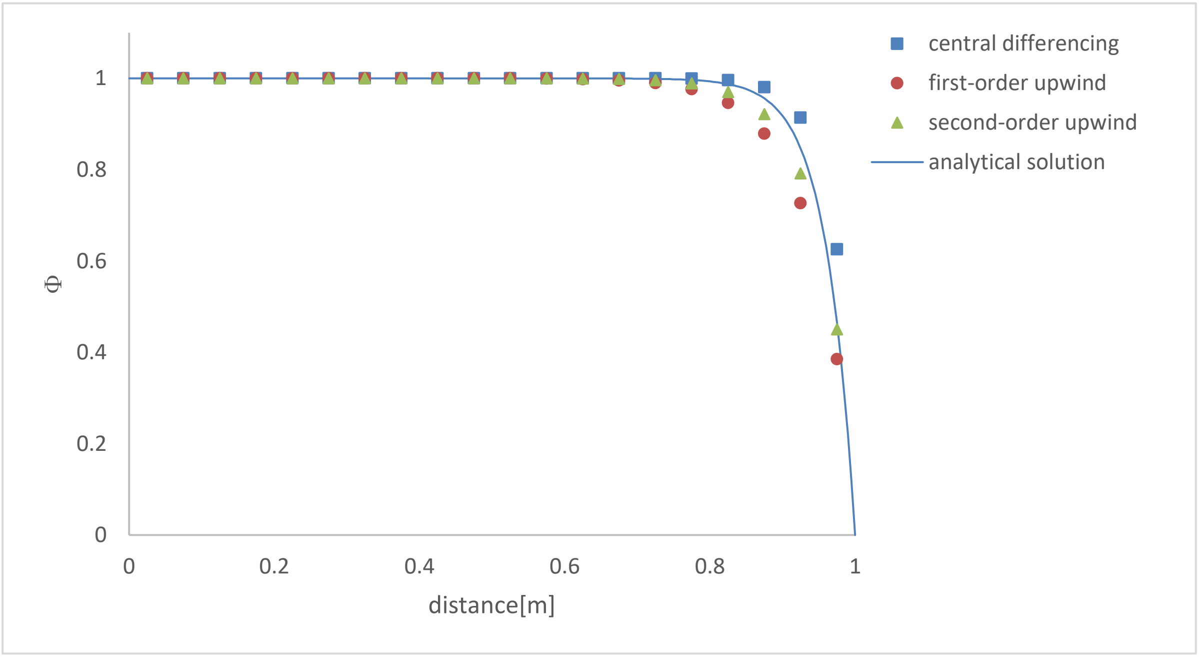

Figure 3.9: Comparision of different schemes

3.6 Solver

Navier stokes equations presents with two problems. First problem is the convective term of the momentum equations contains non-linear quantities such as $div(\rho u\vec{V})$ in Equation (3.2). And another problem is equations are coupled because every velocity component appears in each momentum equation and the continuity equation. Most problematic issue is to solve pressure and there is no equation for the pressure. To resolve the non-linearity and pressure-velocity linkage, iterative solution strategy is needed. In this chapter, a few representative algorithms are presented.

3.6.1 SIMPLE algorithm

SIMPLE stands for Semi-Implicit Method for Pressure-Linked Equations. The algorithm was proposed by Pantankar and Spalding [10]. The algorithm takes a guess-and-correct procedure for the calculation of pressure on the staggered grid. The reason why the staggered grid is used is because if velocities and pressures are stored at the ordinary control volume, the influence of pressure will not be represented in the discretized momentum equations.

The SIMPLE algorithm's work flow is given in Figure 3.10 [11].

A simple example of using SIMPLE algorithm is illustrated below in Figure 3.11. Planar 2-D nozzle is shown with the density of the fluid of $1.0\,kg/m^3$, nozzle length of $2\,m$, inlet area of $0.5\,m^2$, outlet area of $0.1\,m^2$, inlet pressure of 10Pa and outlet pressure of 0Pa. It is solved for velocity and pressure by both using SIMPLE algorithm with upwind scheme and analytically using Bernoulli's equation. The comparison of these two solving methods is given in Figure 3.12 and 3.13. Matlab code is shown in Appendix B-5.

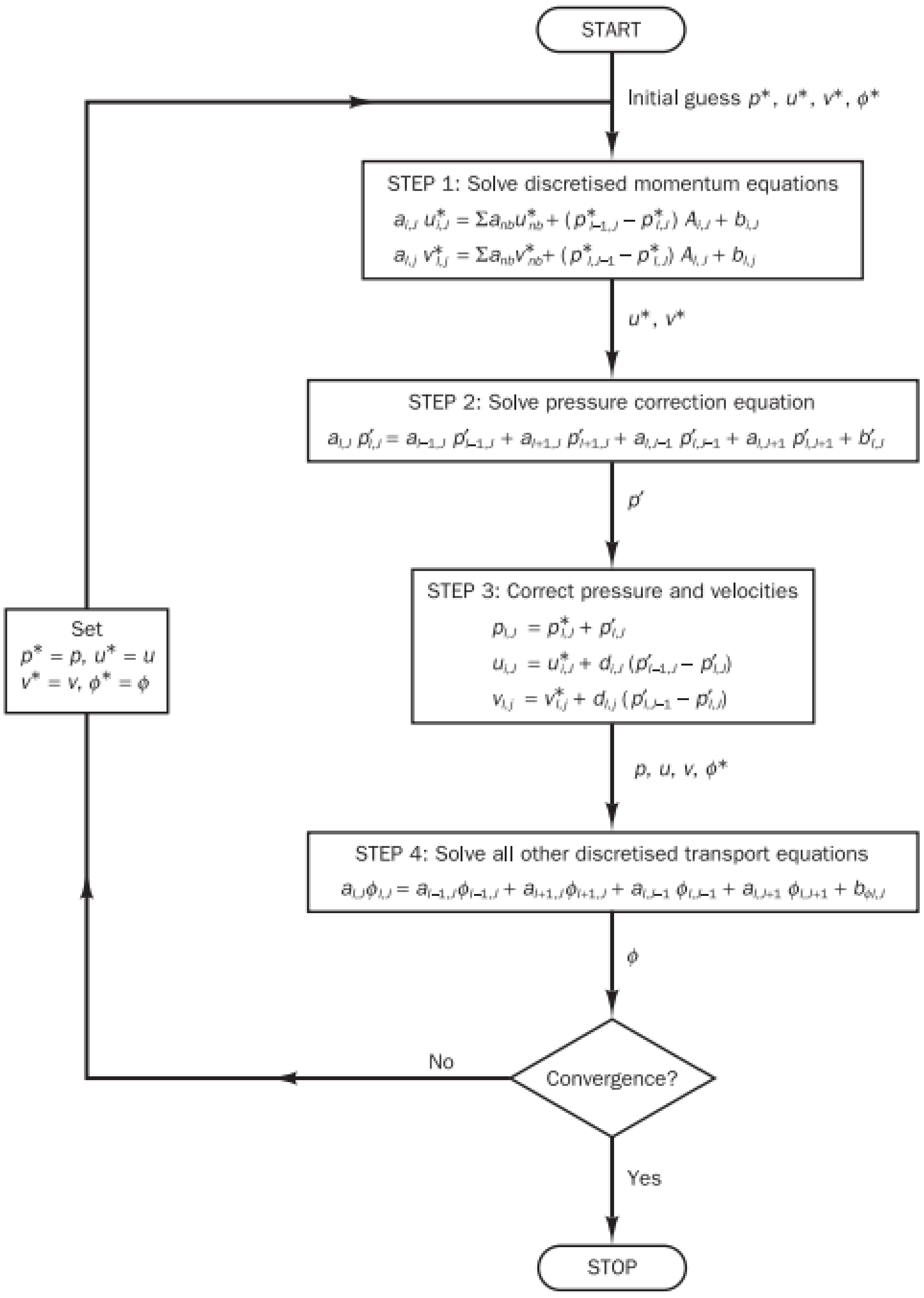
START
Initial guess p^*, u^*, v^*, ϕ^*
STEP 1: Solve discretised momentum equations
$a_{i,J} u^*_{i,J} = \Sigma a_{nb} u^*_{nb} + (p^*_{I-1,J} - p^*_{I,J}) A_{i,J} + b_{i,J}$
$a_{I,j} v^*_{I,j} = \Sigma a_{nb} v^*_{nb} + (p^*_{I,J-1} - p^*_{I,J}) A_{I,j} + b_{I,j}$
u^*, v^*
STEP 2: Solve pressure correction equation
$a_{I,J} p'_{I,J} = a_{I-1,J} p'_{I-1,J} + a_{I+1,J} p'_{I+1,J} + a_{I,J-1} p'_{I,J-1} + a_{I,J+1} p'_{I,J+1} + b'_{I,J}$
p'
STEP 3: Correct pressure and velocities
$p_{I,J} = p^*_{I,J} + p'_{I,J}$
$u_{i,J} = u^*_{i,J} + d_{i,J} (p'_{I-1,J} - p'_{I,J})$
$v_{I,j} = v^*_{I,j} + d_{I,j} (p'_{I,J-1} - p'_{I,J})$
p, u, v, ϕ^*
STEP 4: Solve all other discretised transport equations
$a_{I,J}\phi_{I,J} = a_{I-1,J}\phi_{I-1,J} + a_{I+1,J}\phi_{I+1,J} + a_{I,J-1}\phi_{I,J-1} + a_{I,J+1}\phi_{I,J+1} + b_{\phi I,J}$
ϕ
Convergence?
No
Yes
STOP
Set
$p^* = p$, $u^* = u$
$v^* = v$, $\phi^* = \phi$

Figure 3.10: SIMPLE algorithm

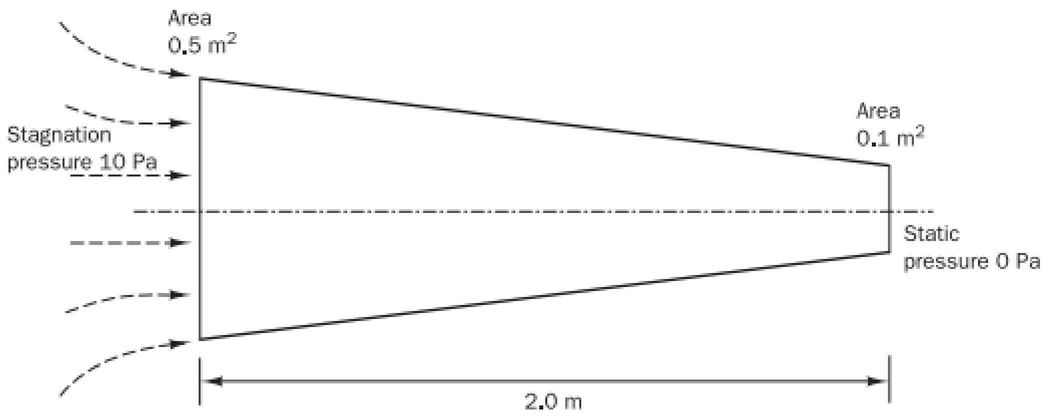

Figure 3.11: Planar 2-D nozzle

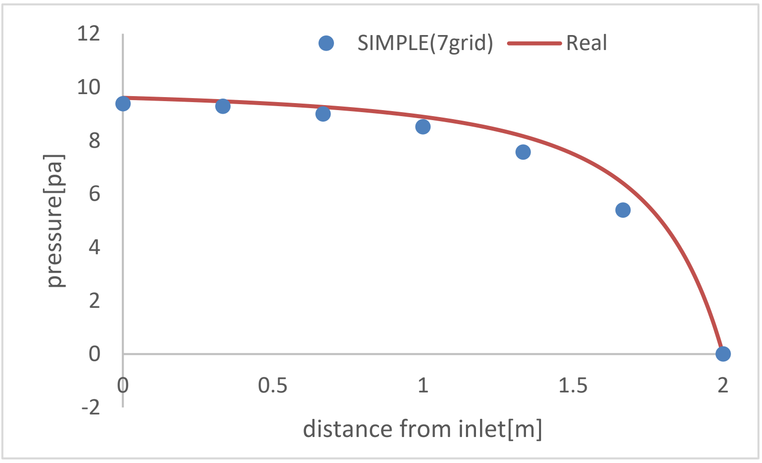

Figure 3.12: Pressure comparision

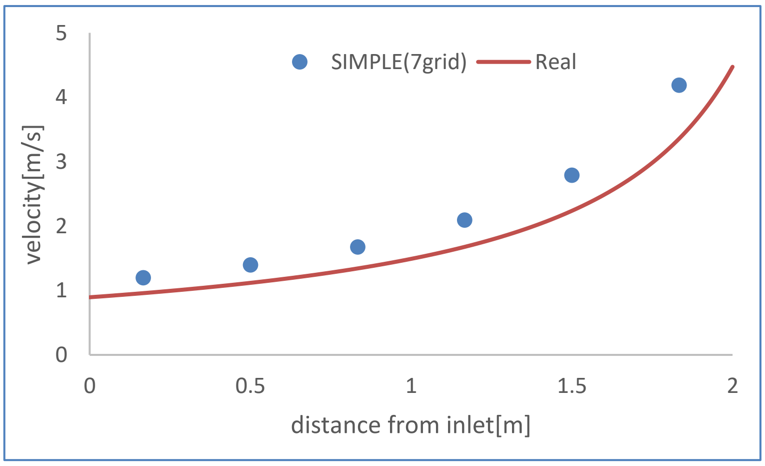

Figure 3.13: Velocity comparision

3.6.2 PISO algorithm

PISO stands for Pressure Implicit with Splitting of Operators, that was proposed by Issa [12]. PISO contains one predictor step and two corrector steps which could be seen as an extension of SIMPLE with a further corrector step. The work flow is shown in Figure 3.14.

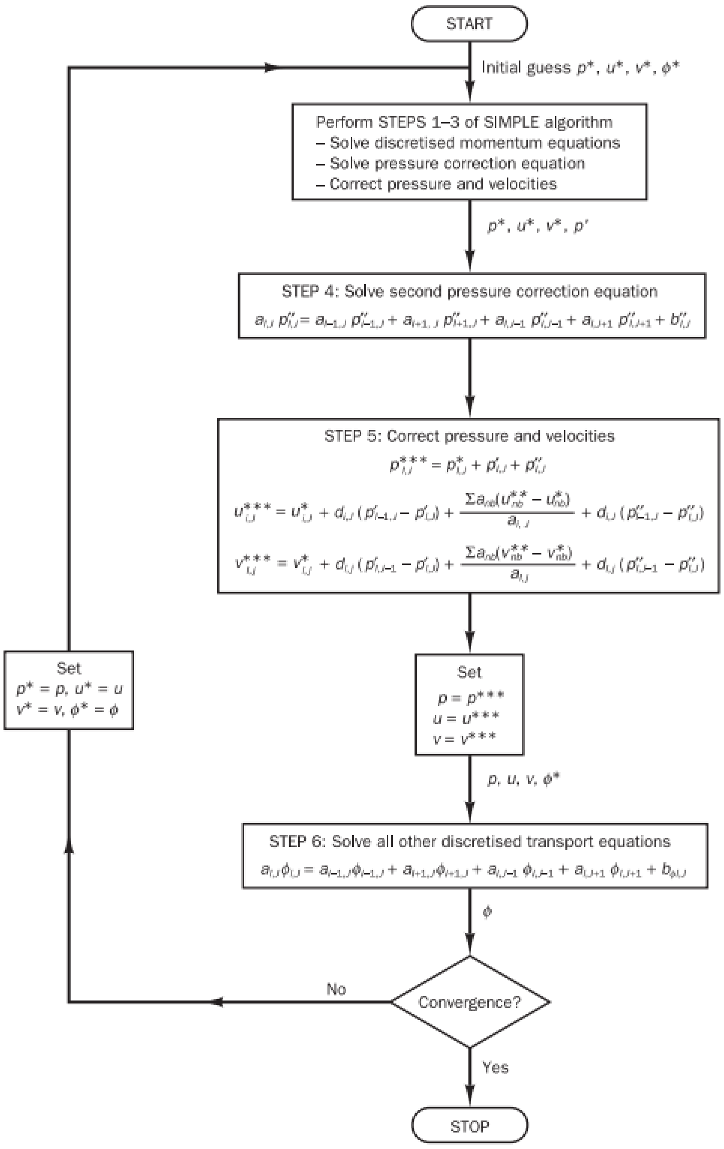
START
Initial guess p^*, u^*, v^*, ϕ^*
Perform STEPS 1–3 of SIMPLE algorithm
– Solve discretised momentum equations
– Solve pressure correction equation
– Correct pressure and velocities
p^*, u^*, v^*, p'
STEP 4: Solve second pressure correction equation
$a_{I,J}\,p''_{I,J} = a_{I-1,J}\,p''_{I-1,J} + a_{I+1,J}\,p''_{I+1,J} + a_{I,J-1}\,p''_{I,J-1} + a_{I,J+1}\,p''_{I,J+1} + b''_{I,J}$
STEP 5: Correct pressure and velocities
$p^{***}_{I,J} = p^*_{I,J} + p'_{I,J} + p''_{I,J}$
$u^{***}_{i,J} = u^*_{i,J} + d_{i,J}(p'_{I-1,J} - p'_{I,J}) + \frac{\Sigma a_{nb}(u^{**}_{nb} - u^*_{nb})}{a_{i,J}} + d_{i,J}(p''_{I-1,J} - p''_{I,J})$
$v^{***}_{I,j} = v^*_{I,j} + d_{I,j}(p'_{I,J-1} - p'_{I,J}) + \frac{\Sigma a_{nb}(v^{**}_{nb} - v^*_{nb})}{a_{I,j}} + d_{I,j}(p''_{I,J-1} - p''_{I,J})$
Set
$p = p^{***}$
$u = u^{***}$
$v = v^{***}$
p, u, v, ϕ^*
STEP 6: Solve all other discretised transport equations
$a_{I,J}\phi_{I,J} = a_{I-1,J}\phi_{I-1,J} + a_{I+1,J}\phi_{I+1,J} + a_{I,J-1}\phi_{I,J-1} + a_{I,J+1}\phi_{I,J+1} + b_{\phi I,J}$
ϕ
Convergence?
No
Yes
STOP
Set
$p^* = p$, $u^* = u$
$v^* = v$, $\phi^* = \phi$

Figure 3.14: PISO algorithm

4 Model Validation

Experiments regarding subsea dispersion were conducted by Engebretsen back in 1997 [3]. The experiments were to investigate the phenomena of the plume and the gas release above the surface. It was performed in fresh water in a rectangular basin of 7m depth and with a surface area of 6 x 9m, with different gas rates of $83\,Nl/s$, $170\,Nl/s$ and 750Nl/s released at the bottom of the basin. Pure air was injected when parameters below the water surface were of interest. And the helium-air mixture was used when gas concentration measurements were to be investigated.

In the further chapters, velocity under the surface of the water, plume's rising time and the fountain that is created by plume are the main focuses. In the experiment, the duration of the air release was done for 20s to eliminate the influence of re-circulating flow in the basin, so it is assumed that the flow reached steady-state for the underwater measurements.

CFD model was introduced by Cloete to compare with the result of Engebretsen's experiment[2]. In the model, DPM and VOF were coupled in ANSYS Fluent, as DPM was used for tracking bubbles and VOF was used for tracking the interface between sea and atmosphere.

In this chapter, Cloete's model is majorly used comparing with Engebretsen's experiment. From chapter 4.1 to 4.5, CFD set-ups based on Cloete's model are presented. In chapter 4.6, the results of simulation and discussion are described.

4.1 Geometry & Mesh

The shape of geometry is a box with the identical size to the basin of the experiment(7m x 6m x 9m). The geometry is created in ANSYS Design Modeler.

The primary mesh is created in ANSYS Meshing with a uniform grid size of 20cm. Then the mesh is refined in ANSYS Fluent. Expected plume area is refined by Region Adaption with level 2. The effect of Region Adaption of a cell is to divide the cell into two cells that have the identical grid size to each other as shown in Figure 4.1.

The total number of cells, faces and nodes are 1130920, 3437126 and 1175723 respectively. Minimum Orthogonal Quality[4] is 8.14553e-01. Maximum Ortho Skew[5] is 1.85447e-01. Maximum Aspect Ratio[6] is 1.74031e+00.

Figure 4.2 describes the exterior of mesh and the mesh on a plane inserted in the middle of the mesh. The mesh of the plane in the middle shows well how the mesh is refined.

[4] Orthogonal Quality ranges from 0 to 1, where values close to 0 correspond to low quality

[5] Ortho Skew ranges from 0 to 1, where values close to 1 correspond to low quality

[6] It is the ratio of longest to the shortest side in a cell. Ideally it should be equal to 1 to ensure best results

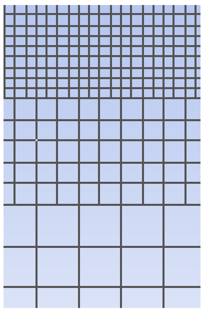

Figure 4.1: Region adaptation

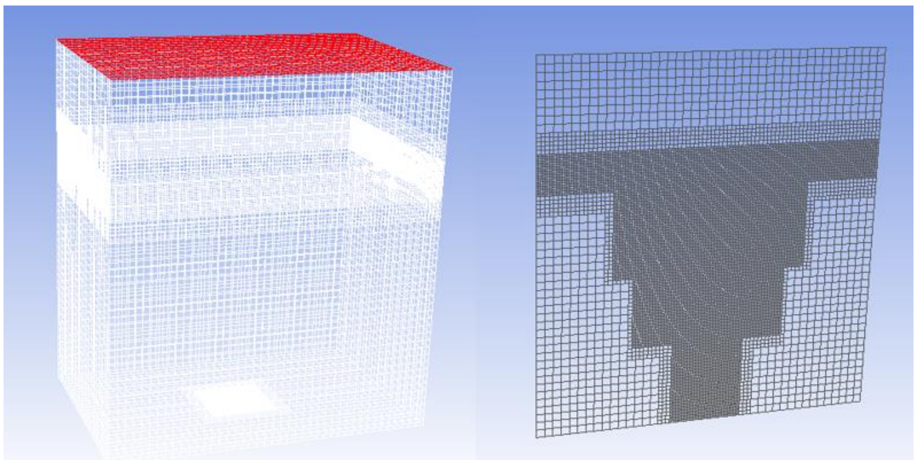

Figure 4.2: Exterior of the mesh on the left side and the mesh on a plane inserted in the middle on the right side

4.2 Boundary & initial conditions

Wall is set as the boundary type for all boundaries, except for pressure-outlet at the top of the mesh. No slip condition is set for wall boundaries. And escape condition is set for the top. This set-up of boundary condition describes that particles (air bubbles) will be reflected when hitting the boundaries that are set as walls while the particles will be removed when hitting the top. In the result, the particles barely reached the wall-boundaries, where almost no reflection of particles is observed.

The model was initialized with zero values for all variables, besides k and ε are guessed to be $k = 0.01 m^2/s^2$ and $\varepsilon = 0.001 m^2/s^3$. These two values are initially distributed in every single cell.

4.3 Models

4.3.1 DPM

Discrete phase model is set with the iteration with continuous phase which accounts for two-way coupling. As a physical model, the virtual mass face was set with the virtual mass factor of 0.5 as a default value.

Injection type of particles was set to be solid-cone type with 0.17m of radius and 13° of cone angle. Particle diameter is set to be 5mm as initial bubble diameter. Parcels were injected via 10 injection streamlines. As a parcel release method, ‘standard’ was used. This method injects a single parcel per injection stream per time step. Therefore 10 parcels were injected every time at 0.33m from the bottom.

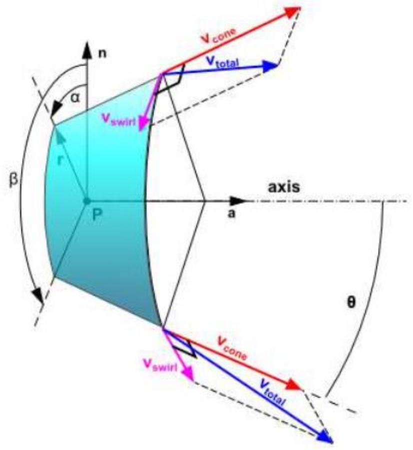

Figure 4.3: Cone injection

For turbulent dispersion, stochastic tracking is used. Unsteady particle tracking is applied.

To find out how Random Walk Model works schematically in the injection, cone injection is modeled in Matlab with a few assumptions to make the model simple. Figure 4.4 shows trajectories of 10 particles only by instantaneous velocity. It represents the main idea of Random Walk. Figure 4.5 shows the trajectories in 2D by mean velocity and instantaneous velocity combined via cone injection with an angle of 13°. The assumption is that trajectory equations(Equation (3.8)) are zero, meaning that drag effect, gravity, interaction with Eurlerian description etc. are neglected, another assumption is that the magnitude of instantaneous velocities is constant as $\sqrt{2k/3}$. Figure 4.5 on the right-hand side, describes the effect of k value. The code is attached in Appendix B-6.

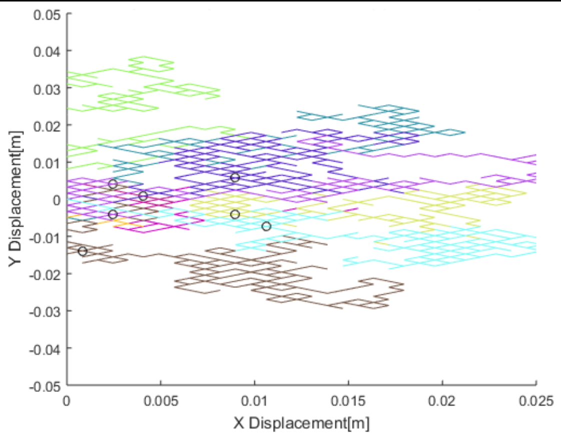

Figure 4.4: Ttrajectories by instantneous velocity

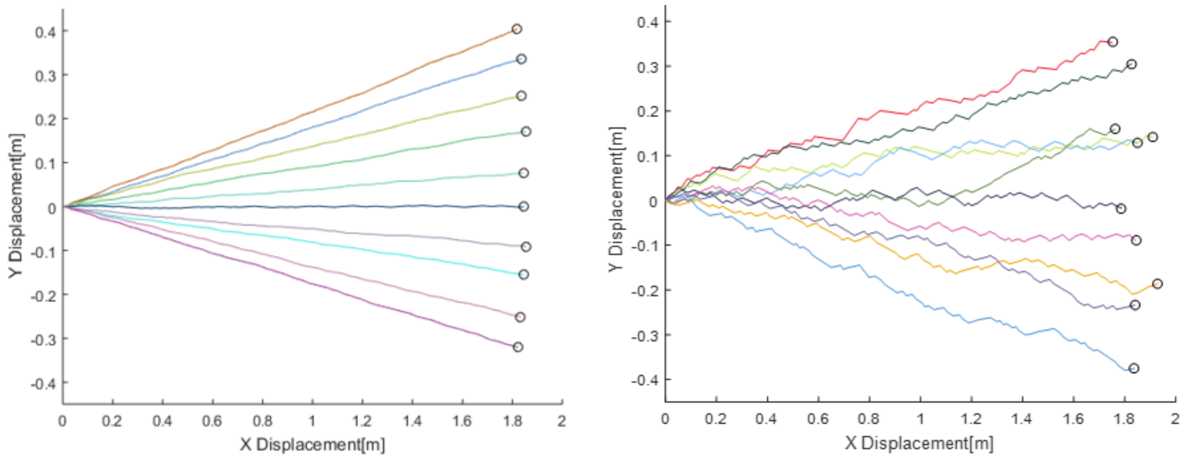

Figure 4.5: 2D cone injection with different k value. Left one with k=0.01, right one with k=0.5

As for drag, mainly Xia’s drag is used as the drag of the original model by Cloete. Additionally, the Tomiyama’s drag, Spherical drag and the Modified Spherical drag that are mentioned in the previous chapter are used. The results of different drags are shown in chapter 4.6.2.

4.3.2 VOF

For multiphase model, Volume of Fluid is used with the formulation of explicit and the interface modeling of sharp. The density and viscosity of the water phase are set to 998.2 kg/m^3 and 1.003e-3 $kg/m \cdot s$ respectively. For air, it was given as 1.225 kg/m^3 and 1.7894e-5 $kg/m \cdot s$.

4.3.3 Turbulent model

Standard k-epsilon model is used with default model constants.

4.3.4 User defined model

For the sake of implementing the change of bubble's density and size, user-defined modeling is needed. Otherwise ANSYS Fluent recognize particles with a constant size and density which do not account for phenomena of bubble such as coalescence and break-up. Basic theories of bubble density change and bubble size model that were implemented are introduced in the following. The UDF of the user defined model was provided by Mikkel[9].

4.3.4.1 Bubble density change

The bubble is considered as ideal gas. Therefore, the density of bubble(gas) is defined as

$$\rho_g = \frac{P_{hd}}{R_g T_{fl}} \tag{4.1}$$

Where P_{hd} is hydrostatic pressure, R_g is gas constant and T_{fl} is the temperature of the ambient fluid. T_{fl} was assumed to be constant which is reasonable since the user defined model was used for shallow depth cases. The temperature change is not significant in shallow depth as shown in Figure 4.6**.**

Hydrostatic pressure is the pressure that presents within a fluid when it is at rest. It acts equally in all directions. Also, it acts to any surface in contact with the fluid perpendicularly. It is well described in Figure 4.7.

The pressure change by hydrostatic pressure is defined as

$$\Delta P = \rho g \Delta h \tag{4.2}$$

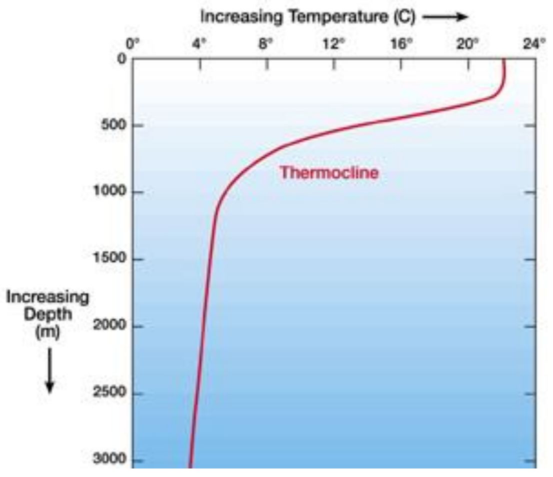

Figure 4.6: Temperature change under water

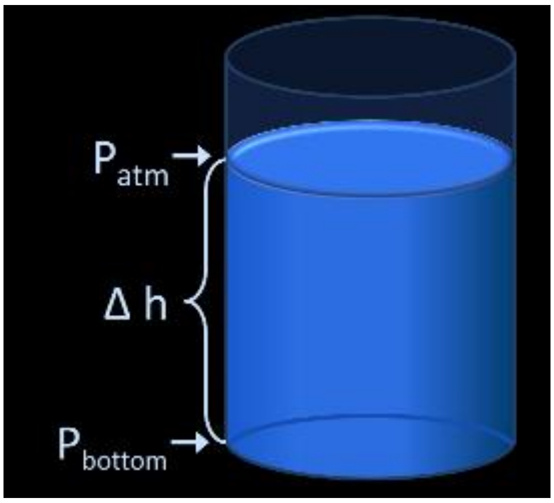

Figure 4.7: Hydrostatic pressure

4.3.4.2 Bubble size model

The bubble size model is controlled by material properties and turbulence. Local mean bubble diameter (d_p) accounts for loss of bubbles to downstream cells, gain of bubbles from upstream cells, break-up and coalescence [2]. In Lagrangian framework, it is defined such,

$$\frac{\partial \rho_p d_p}{\partial t} = \rho_p \frac{d_p^{eq} - d_p}{\tau} \tag{4.3}$$

Where ρ_p is the bulk density, τ is the relaxation time and d_p^{eq} is the mean equilibrium diameter. The equilibrium diameter is the diameter that it is achieved if a bubble resides long enough at the same flow conditions. It is defined such

$$d_p{}^{eq} = C_1 \phi_p^{0.5} \frac{(\sigma / \rho)^{0.6}}{\varepsilon^{0.4}} \left(\frac{\mu_p}{\mu} \right)^{0.25} + C_2 \tag{4.4}$$

Where the coefficients C_1 and C_2 are 4 and $100\mu m$ respectively.

The relaxation time is the time that is needed for bubble to reach the equilibrium diameter.

The mean bubble diameter will be driven to its equilibrium diameter during a timeframe given by the relaxation time.

The relaxation times for breakup (τ_{br}) and coalescence (τ_{co}) are given by the turbulence dissipation rate and kinetic energy respectively. They are defined such

$$\tau_{br} = d_p^{2/3} \varepsilon^{-1/3}, \;\; \tau_{co} = d_p / (0.2 * 6 * \sqrt{\phi_p k}) \tag{4.5}$$

When the model is implemented in code, if a bubble is bigger than $d_p{}^{eq}$ then the bubble's breakup occurs otherwise the coalescence will occur. τ_{br} or τ_{co} will be obtained, the relaxation time is restricted by turbulent microscale that represents the smallest timescale in turbulent flow [13]. Bubble size is restricted to have a diameter size above 0.0001m. The fraction of the bubble is also restricted to be below $1.0e-06$.

4.4 Solver & scheme

For solver, PISO algorithm is used. For gradient scheme, least square cell based is used. PRESTO! for pressure, Geo-Reconstruct for volume fraction, second order upwind for momentum, turbulent kinetic energy and turbulent dissipation rate and first order implicit for transient formulation are used as shown in Table 4.1.

The timestep is set to 0.01s with the maximum 100 iterations per time step. Simulation is executed till 20s simulation time as same as the duration of the release in the experiment.

The limit of the residuals are set to 1e-05 for every variable. The under-relaxation factors are set to the default value.

Table 4.1: Scheme

Gradient	least square cell based
Pressure	PRESTO!
Momentum	second order upwind
Volume fraction	Geo-Reconstruct
Turbulent kinetic energy	Second order upwind
Turbulent dissipation rate	Second order upwind
Transient formulation	First order implicit

4.5 Cases

Three different comparisons were done for the Engebretsen’s experiment. In Case 1, a CFD model is validated by using Cloete’s model.

Based on the basic CFD model by Cloete, different drags are implemented to see their influences in Case 2.

Case 3 is to compare the integral model that is introduced in chapter 1.2 with the results of the experiment and the simulation.

Table 4.2 shows the overview of the three cases.

Table 4.2: Cases

	Comparison	Drag
Case 1	Validation of Cloete's CFD model	Xia's drag
Case 2	Effect of different drags	Spherical drag law Modified Spherical Drag law Tomiyama's drag law
Case 3	Comparison between Integral model, CFD model and experiment	

4.6 Result & Discussion

4.6.1 Case 1

Fountain is observed proving that DPM and VOF are coupled as shown in Figure 4.8. Flow reflection is observed as shown in Figure 4.9. Maximum, minimum and mean diameter of bubble are reported 7.96e-02m, 1.67e-03m and 5.32e-03m respectively. Bubble parcels barely touch the wall boundaries. Parcels are removed when touching the surface of the water by UDF implemented. Almost no change in under water measurements is observed after 20s of simulation, which can be assumed to be steady-state.

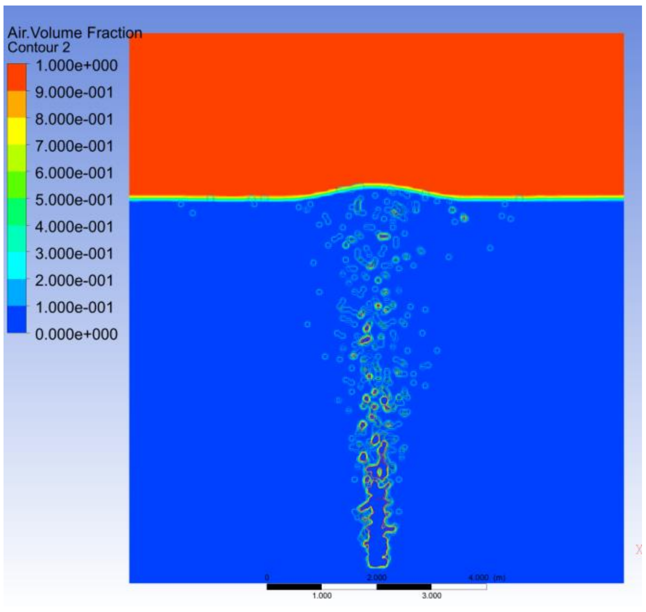

Figure 4.8: Fountain

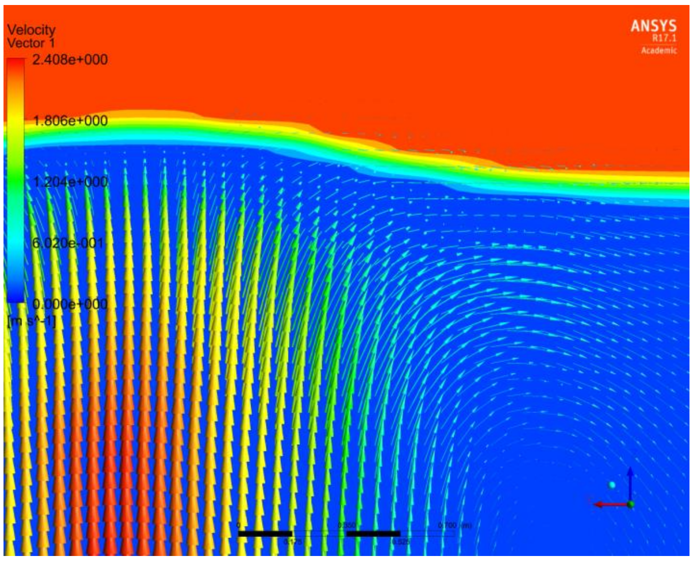

Figure 4.9: Flow reflection

4.6.1.1 Velocity

Velocities at the flow rate of 170 Nl/s is of interest at the vertical positions(z-direction) of 1.75m, 3.80m and 5.88m at 20s. Velocity profiles at 1.75m and 3.80m depth of the simulation match the experiments well. In the meantime, quite large deviation at 5.88m is observed. This is due to the unsuitable use of the meter for measuring multi-directional flow during the experiment[14], meaning that as a flow approaches the surface of the water, the flow becomes a multi-direction flow in which radial flow is predominant. However, the meter that is used suitable for mono-directional flows. This explains why the velocity only around the center at z=5.88 where the vertical flow is predominant shows a good match. Therefore, in the further comparisons, the velocity profile at z=5.88m is not considered for validation. The velocity profiles are shown in Figure 4.10.

In addition to the velocity measurement at three different z positions, the velocity near the surface is measured at 1.75m from the plume center for the gas flow rates of 83 Nl/s and 170 Nl/s. The comparison between the experiment and simulation is shown in Figure 4.11.

The simulation results have the most deviation at the flow rate of 170 Nl/s, especially at the nearest points from the surface. It is proved that the model with surface damping correction gives improved results[14].

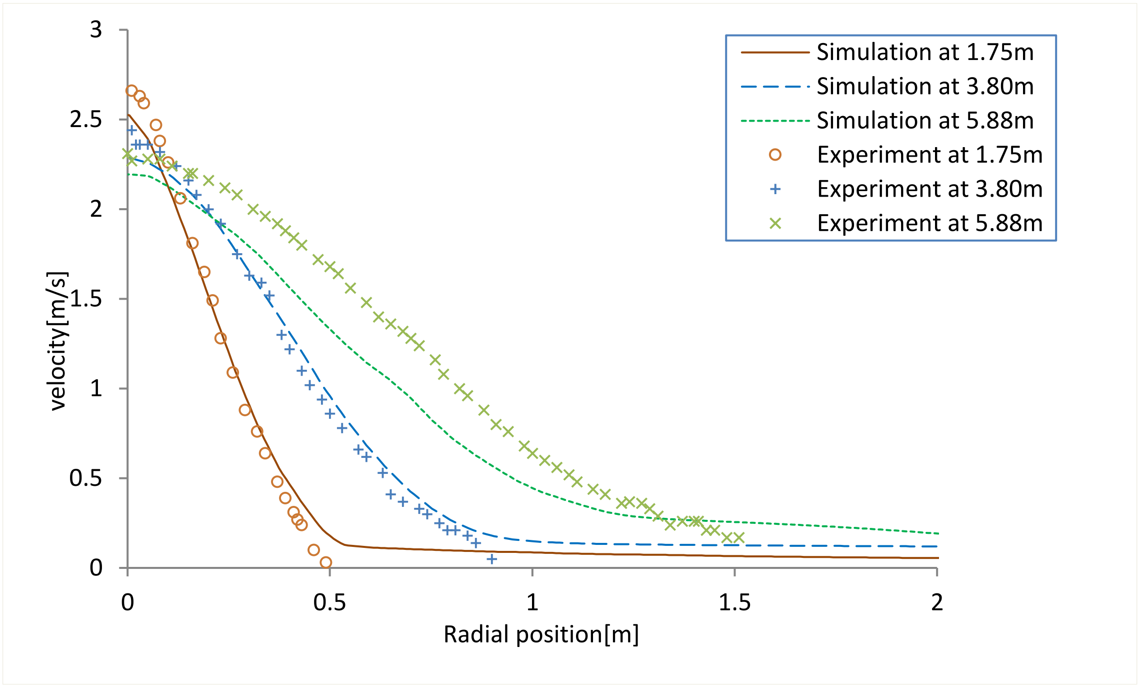

Figure 4.10: Velocity profile at z=1.75, 3.80 and 5.88

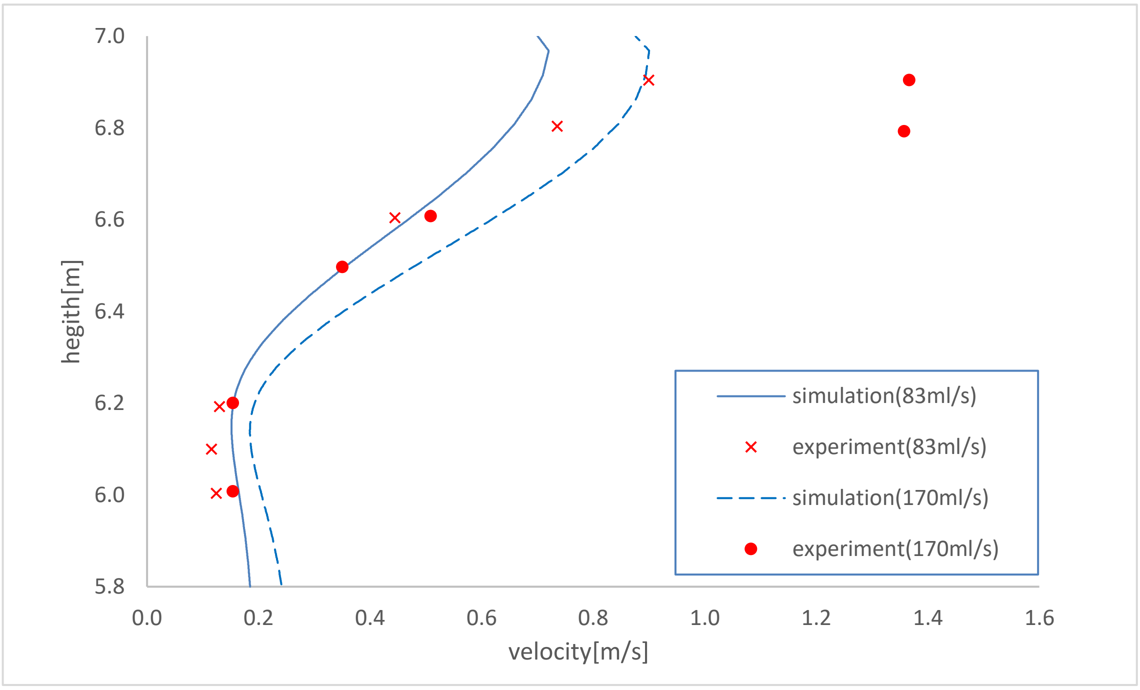

Figure 4.11: Velocity near the surface

4.6.1.2 Height of fountain and rising time

In Engebretsen’s experiments, the initial fountain height when the plume reached the surface for the first time and the maximum fountain height were measured. The comparison between the experiment and simulation is shown in Table 4.3.

Table 4.3: Foundtain height and plume rising time

	Experiment			Simulation		
Flow rate [Nl/s]	83	170	750	83	170	750
Plume rising time [s]	6	4.8	3.1	7.2	5.5	3.25
Initial fountain height [m]	-	0.3	0.45	-	0.18	0.29
Maximum fountain height [m]	-	0.65	1.25	-	0.43	0.87

Heights of fountain seem to have considerable deviations. In the experiment, it is mentioned that one of equipment was splashed at 2 meters above from the water surface. Meanwhile, the simulation does not really show the huge motion of splash unlike the experiment.

Slightly over predicted values are observed in rising time. It is reported by Mikkel[9] that higher initial turbulent kinetic energy(k) values cause a longer rise time. Also, it is mentioned that variables that are relevant to k might cause more dispersed flow. To investigate how initial k affects the rising time, simulations with the different initial k values of 0.007 m^2/s^2 and 0.014 m^2/s^2 are done.

Plumes at 5s are captured as shown in Figure 4.12. Significant difference of plume position is oberved at this time.

Visible difference in k in plume that have been developed by the time is observed in contour images as shown in Figure 4.13. Fixed range scale is used for both cases to see the difference.

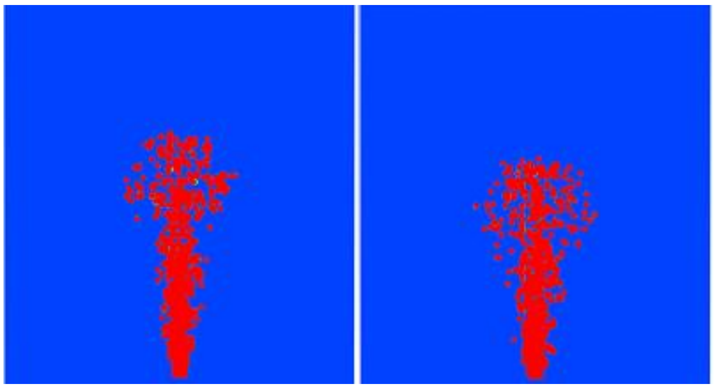

Figure 4.12: Plume at 5 s

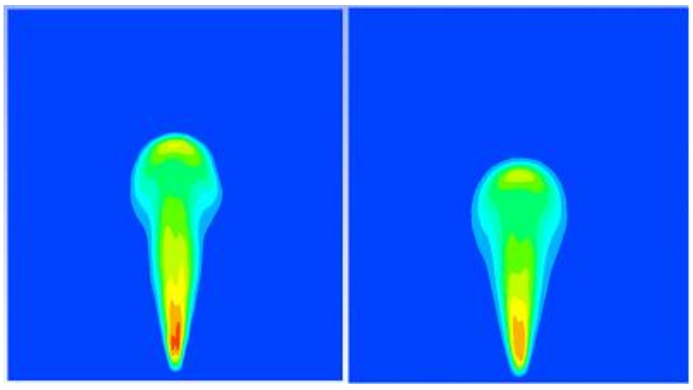

Figure 4.13: Contour of k at 5s

Also k value in plume is measured at the center line of the plume. It is shown in Figure 4.14. From the measured k values, it is obvious that the plume in the field of the smaller initial k value of 0.007 m^2/s^2 propagates faster, also the plume has a higher k value that is devloped over time. Therefore it can be assumed that lower initial k values in the mesh field result in higher k values of plume. And higher k values in plume contribute to faster rising velocities.

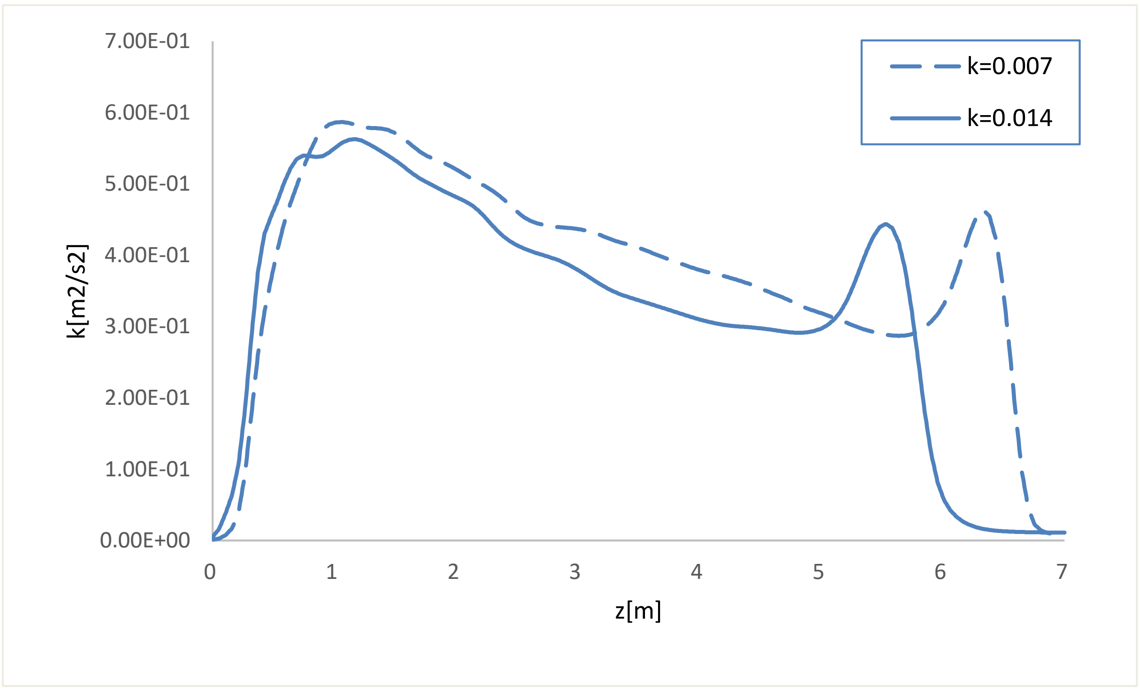

Figure 4.14: Measured k value at the center line of plume

4.6.1.3 Void fraction

Since the volume fraction of particles cannot be obtained in ANSYS Fluent without time sampling, it is measured by time sampled data of every second for 20 seconds. RMS value is measured. The result shows a quite large deviation as described in Figure 4.15. The void fraction ranges from 0.3 to 0.5. Even larger deviation is observed when Mean value is used. According to Fluent manual, the rule of thumb of using DPM is that volume fraction of particles should be lower than 10-12%. This represents the assumption that the particle-particle interations and the effects of the particle volume fraction on the gas phase are negligible. In that regard, the violation of the recommended range was observed. It might have possibly caused errors in any results.

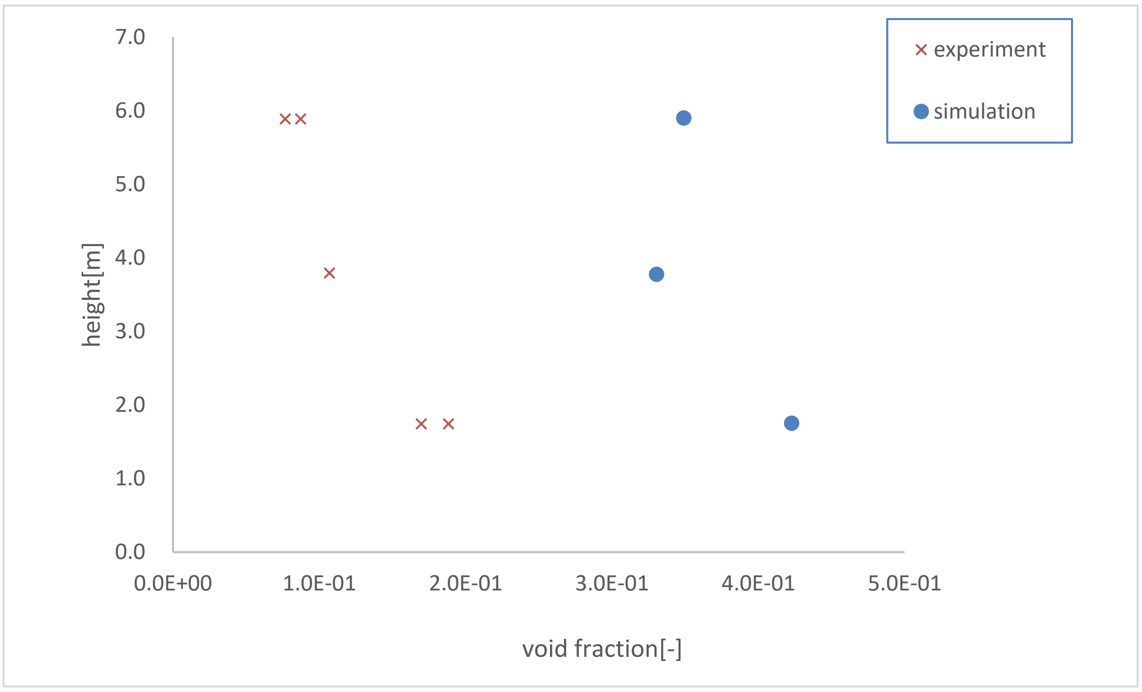

Figure 4.15: Void fraction

4.6.2 Case 2

To see how drag can affect the result, three different drags from chapter 3.4.1.4 are implemented. Spherical drag and Modified spherical drag do not account for the shape of bubble but only Reynolds number. Meanwhile, Tomiyama's drag and Xia's drag account for the shape of bubble. The influence of including a factor of the bubble shape is expected to be shown in this chapter. Simulations are done only for the flow rate of 170 Nl/s. And none of other set-up values are changed but only drag.

4.6.2.1 Velocity

Velocity profiles of spherical drag and the modified drag seem to match well the experiment data, except for the center area of the plume. In the center area, larger deviation than the original model that is presented in case 1 (with Xia's drag) is observed. Spherical drag and Modified drag seem to give almost the same result except for the fact that Spherical drag results in a better match by a bit within the range of radial position from 0 to 0.5m.

Reynolds number of Spherical drag coefficient is divided into seven ranges from Re= 0 to 10000 while it is divided into three ranges in Modified spherical drag coefficient. Judging from different division of Re, it can be assumed that Spherical drag gives more accurate result in lower Re, while Modified spherical drag give accurate result in higher Re. Therfore it is reasonable to assume the better result within the range of radial position from 0 to 0.5m might have been caused by plume's low Re.

Meanwhile the simulation result from Tomiyama's drag seems to match the best among 3 drags, even at around the center area. And especially at z=3.8m, the result matches the experiment data better than Xia's drag. The results with different drags are shown in Figure 4.16-4.19. The reason why Tomiyama's drag gives a better result than Xia's can be assumed that Reynolds number also is included in Tomiyama's drag coefficient, not only the shape of bubble.

Velocities of the different drags near the surface are also compared. It is shown in Figure 4.20. Disregarding the damping effect which is not applied in the model, the results of velocity near the surface show less deviation from the experimental data when Spherical drag and Modified spherical drag are used. When Tomiyama's drag is used, the result is almost identical to the result by Xia's drag.

It is observed that the drags that include the shape of bubble give a better result besides the velocity near the water surface. Assuming bubble's shape is a crucial factor in plume seems to be reasonable. However, since the drags that including only Reynolds number result in a better match near the water surface, it might be reasonable to assume that Reynolds number is more important than the shape of bubble near the surface.

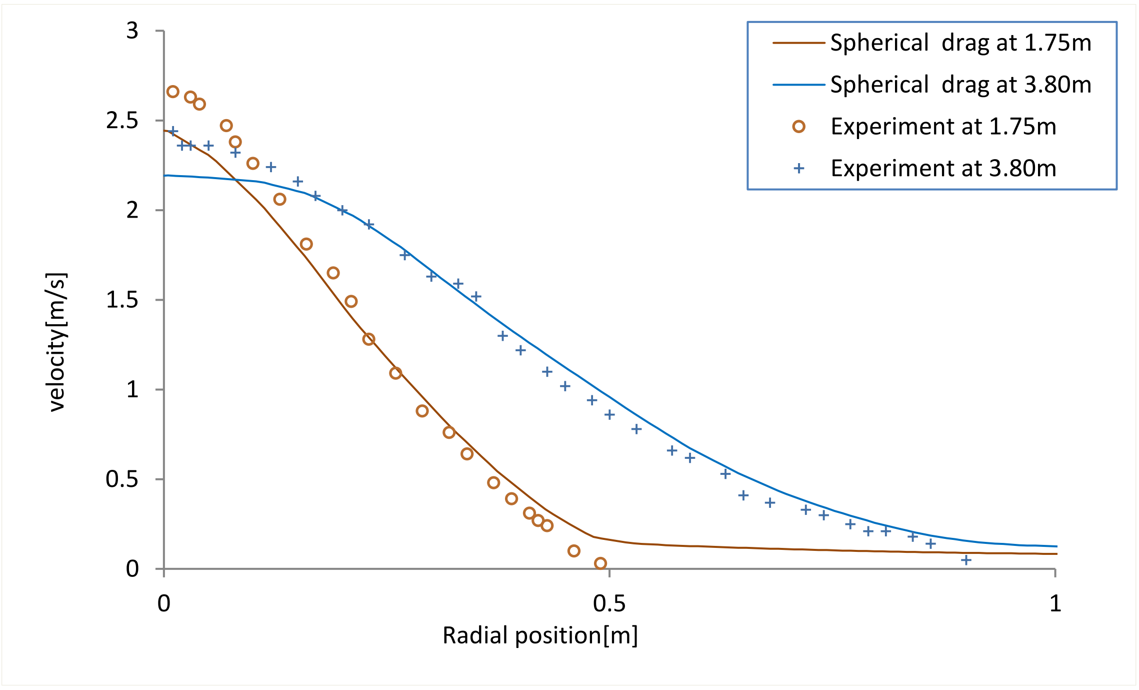

Figure 4.16: Velocity by Spherical drag

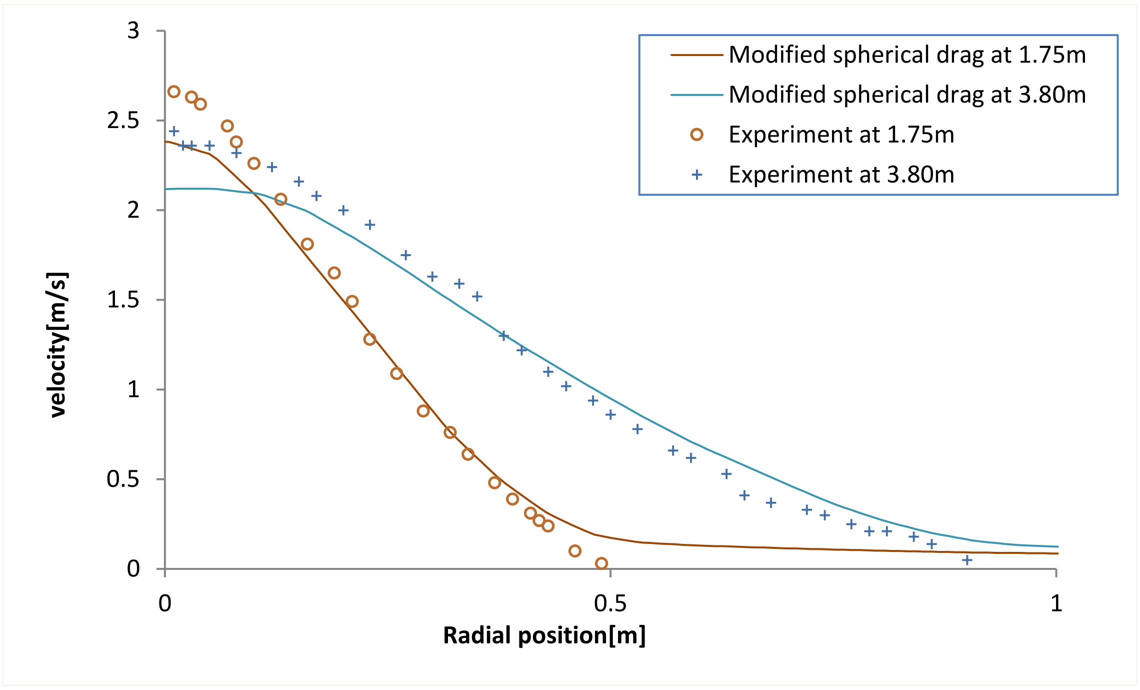

Figure 4.17: Velocity by Modified spherical drag

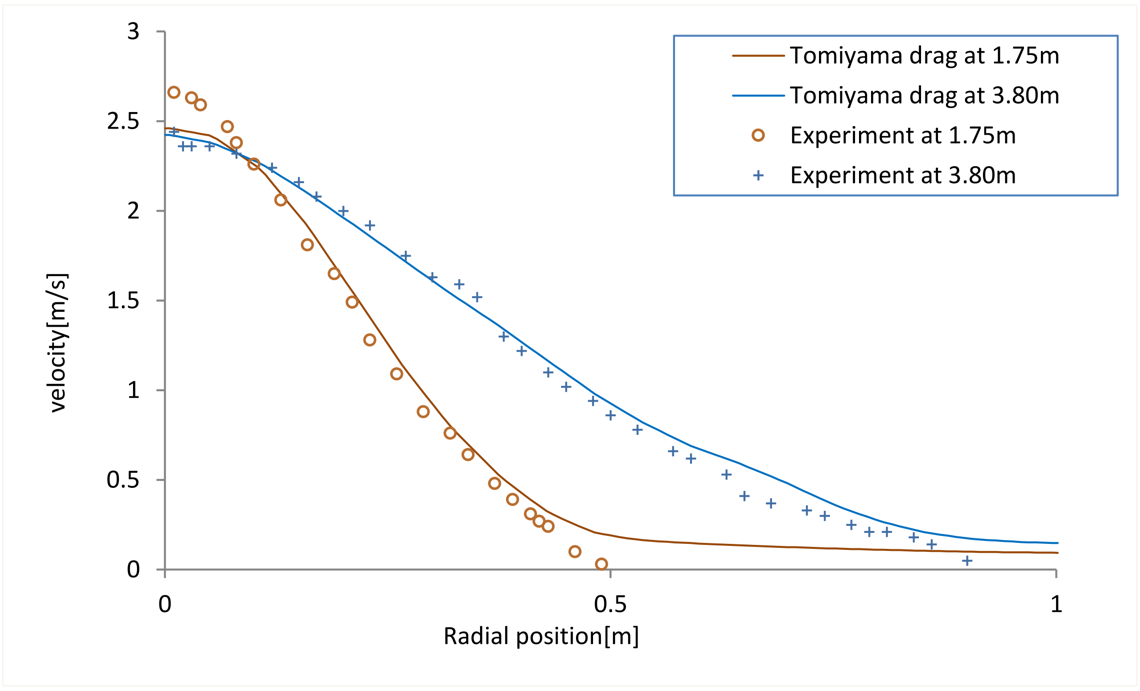

Figure 4.18: Velocity by Tomiyama’s drag

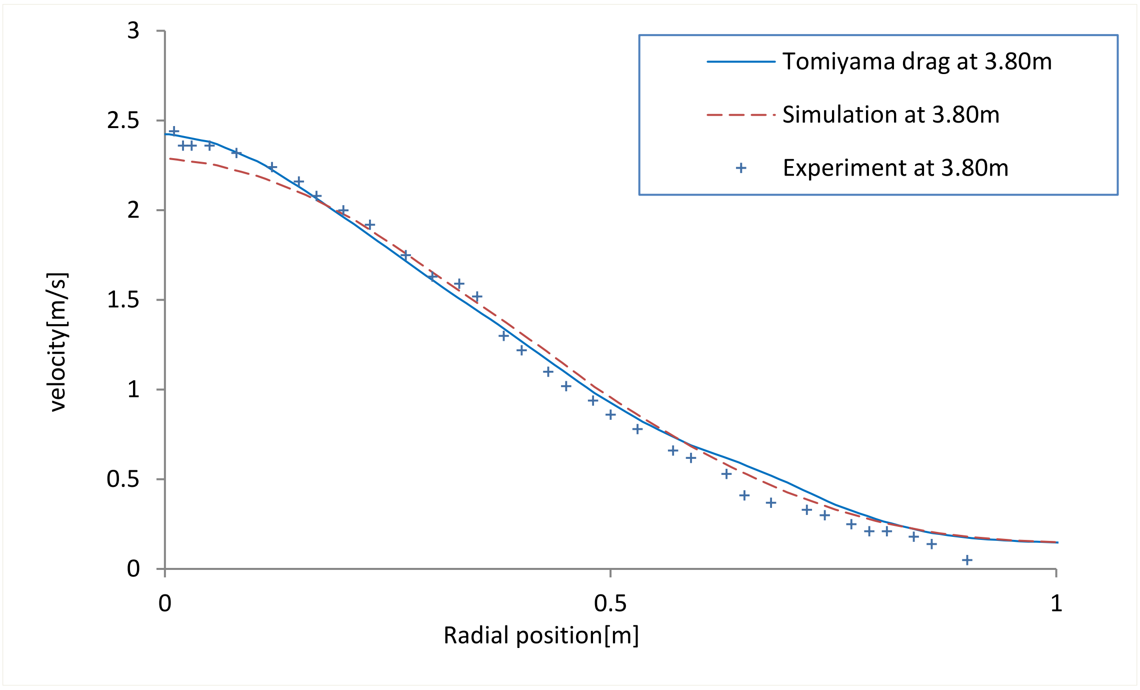

Figure 4.19: Comparison between the results by Tomiyam's drag, Xia's drag and experiment

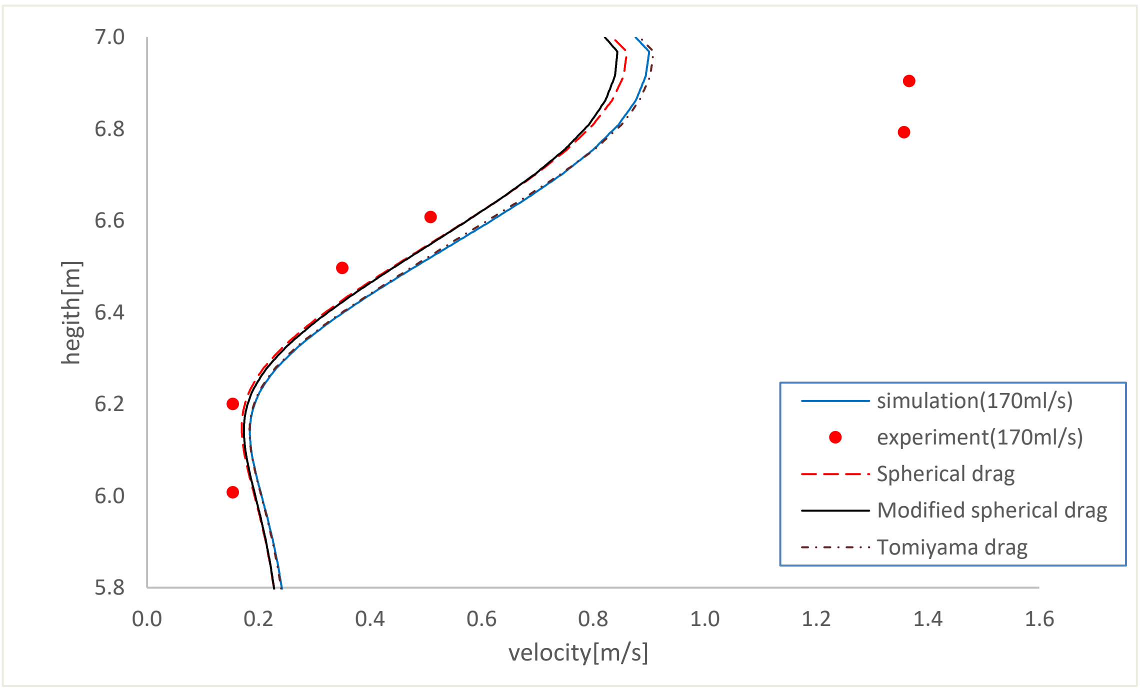

Figure 4.20: Velocity by different drags near the water surface

4.6.2.2 Height of fountain and rising time

Spherical drag and Modified spherical drag again show the almost identical results in plume rising time and fountain height. And rising time from the drags shows only a little deviation from the experimental data. Xia's drag and Tomiyama's drag give the similar results besides the maximum fountain height.

Good agreement of the fountain height is not observed in any of the simulations comparing with experimental data. The overview of comparison is shown in Table 4.4.

As mentioned in chapter 4.6.1, plume rising time is rather dependent upon initial k value. Therefore, it is reasonable to say that initial k =0.01 m^2/s^2 is a good guess for both Spherical and Modified spherical drag.

Table 4.4: Plume rising time and fountain height by different drags

	Experiment	Xia's drag	Spherical drag	Modified spherical drag	Tomiyama's drag
Flow rate [Nl/s]	170	170	170	170	170
Plume rising time [s]	4.8	5.5	4.6	4.75	5.5
Initial fountain height [m]	0.3	0.18	0.13	0.13	0.19
Maximum fountain height [m]	0.65	0.43	0.34	0.34	0.35

4.6.3 Case 3

The integral model is investigated for the flow rate of 170 Nl/s. The comparisons between the integral model, the CFD model and the experimental data are seen in case 3.

4.6.3.1 Velocity

Integral model is determined by tuning the coefficients that are introduced in chapter 2.2. In this thesis, only entrainment coefficient α and proportionality value γ are manipulated.

γ is observed to determine the peak of velocity profile and α is observed to affect the entire shape such as the peak velocity in the center and width of the velocity profile. Comparisons by manipulation of γ and α are shown in Figure 4.21 and 4.22.

For the velocity at the height of z=1.75m, the curve of the integral model seems to be optimized when $\alpha = 0.1285$ and γ =1.5 are used. And for the velocity at z=3.80m, using $\alpha = 0.11$ and γ =1.5 showed a good match. Also, both tunings showed good matches to experiments over the velocities of the vertical direction. However, the velocity at the low z position diverges infinitely, it is due to the calculation of the approximate solutions where 0 is put into denominator. The data at small z position should not be considered as valid value. The results are shown Figure 4.23~4.25.

The biggest deviation is observed in the near velocity comparison as shown in Figure 4.26. This is due to the assumption of the integral model that the velocity of plume follows Gaussian profile. Therefore, the flow reflection near the surface cannot be explained in the integral model.

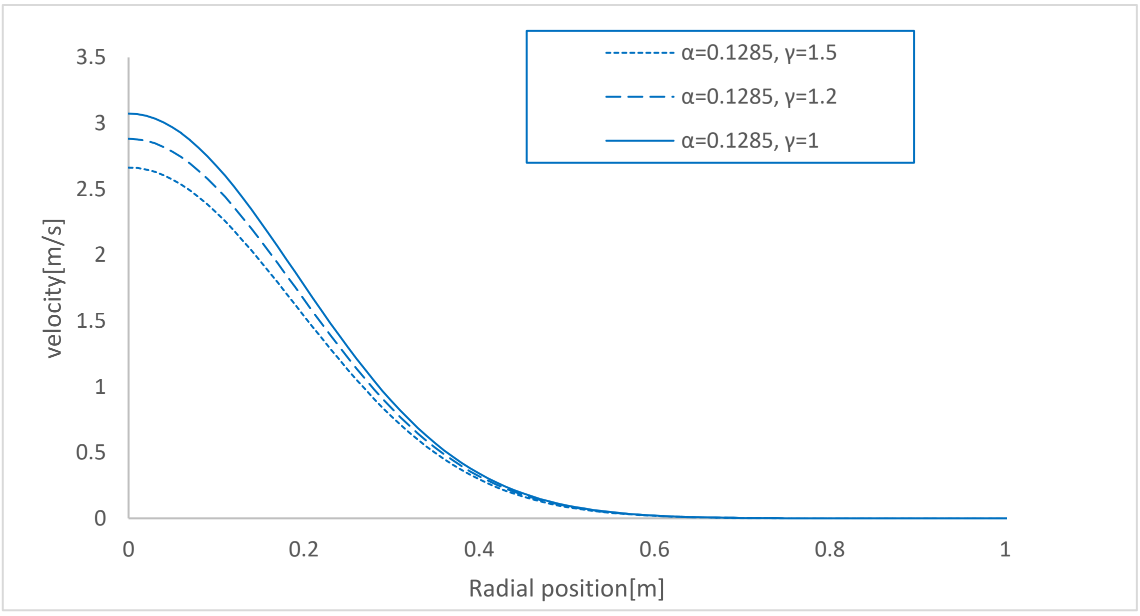

Figure 4.21: γ manipulation

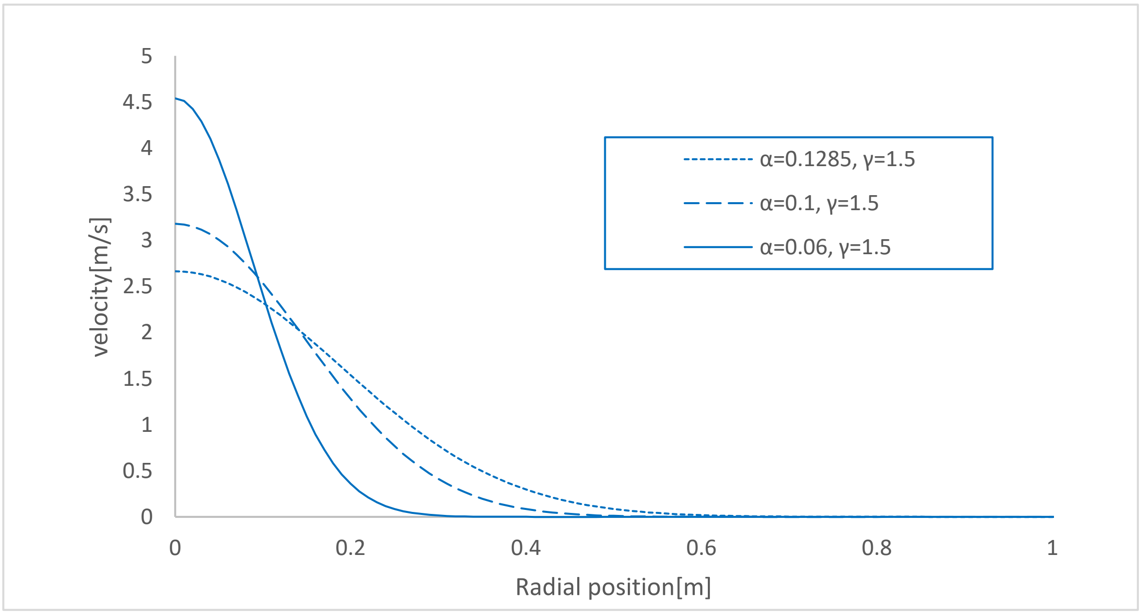

Figure 4.22: α manipulation

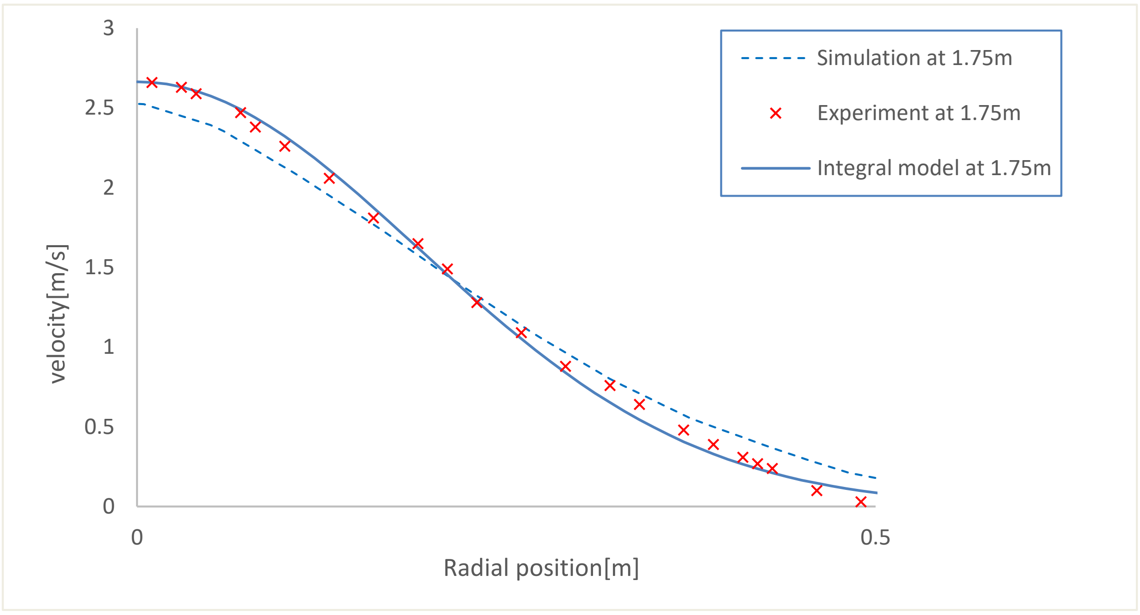

Figure 4.23: Velocity at z=1.75 when α = 0.1285 and γ =1.5

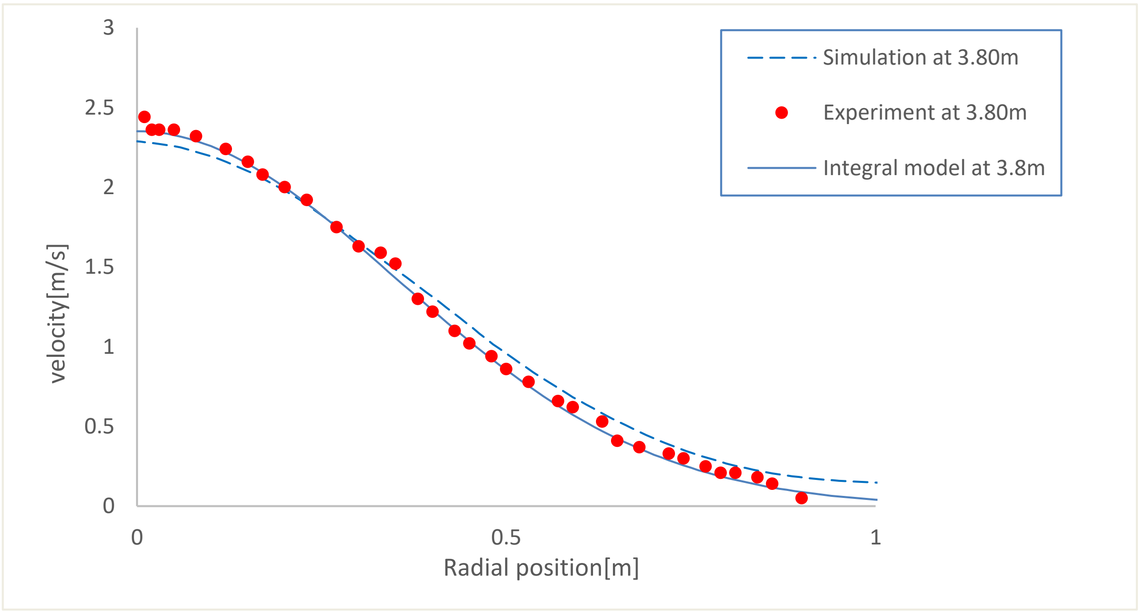

Figure 4.24: Velocity at z=3.80m, using α = 0.11 and γ =1.5

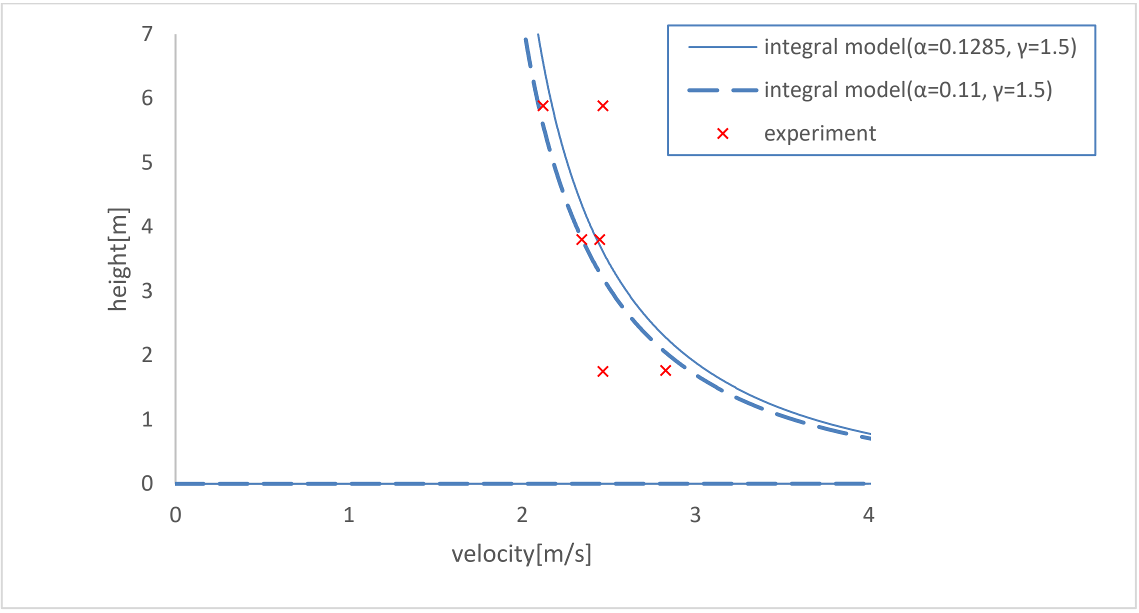

Figure 4.25: Comparison between two manipulations

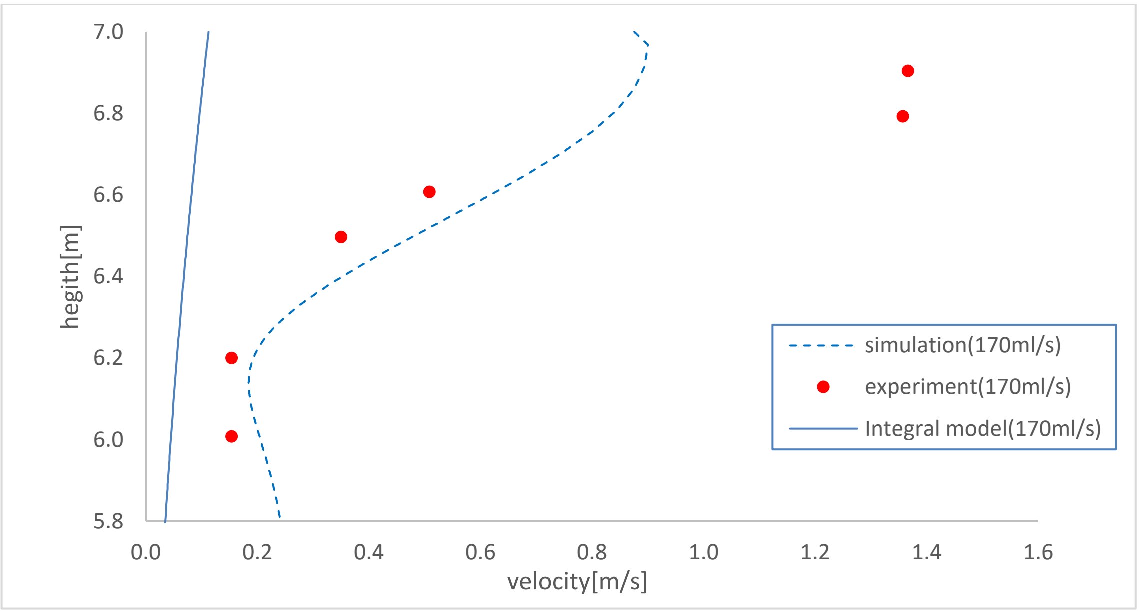

Figure 4.26: Velocity near the surface

4.6.3.2 Height of fountain

One of the parameters in the integral model that hugely impacts the height of the fountain is empirical constant β. It accounts for the loss in momentum balance. For a free-loss rise $\beta = 0.5$. However, Friedl[4] mentioned the empirical constant $\beta \approx 0.39$ seems to be the best value determined through a fitting procedure with experimental data. Therefore, $\beta = 0.39$ is used for the comparison of fountain height.

In addition, Friedl provided relations of the average peak of instantaneous height $\overline{h}_p$ and the maximum peak of instantaneous height $h_{p\max}$ with respect to the mean surface elevation h_f.

$$\overline{h}_p = 2h_f, \quad h_{p\max} = 3.1h_f \tag{4.6}$$

The values following above equation are $\overline{h}_p = 0.39m$ and $h_{p\max} = 0.6m$ when α = 0.1285 and γ =1.5. Since the average fountain height of the experiment was not provided by Engebretsen, $\overline{h}_p$ cannot be compared directly. However, if it is assumed that the fountain heights after initial height would be the values between the initial height and the maximum height, then the heights from the integral model seem to give a good result with a small error.

4.6.3.3 Void fraction

Void fraction is also calculated using α = 0.1285 and γ =1.5. The result is shown in Figure 4.27. Compared to the CFD simulation, the result of the integral model matches the experimental data well.

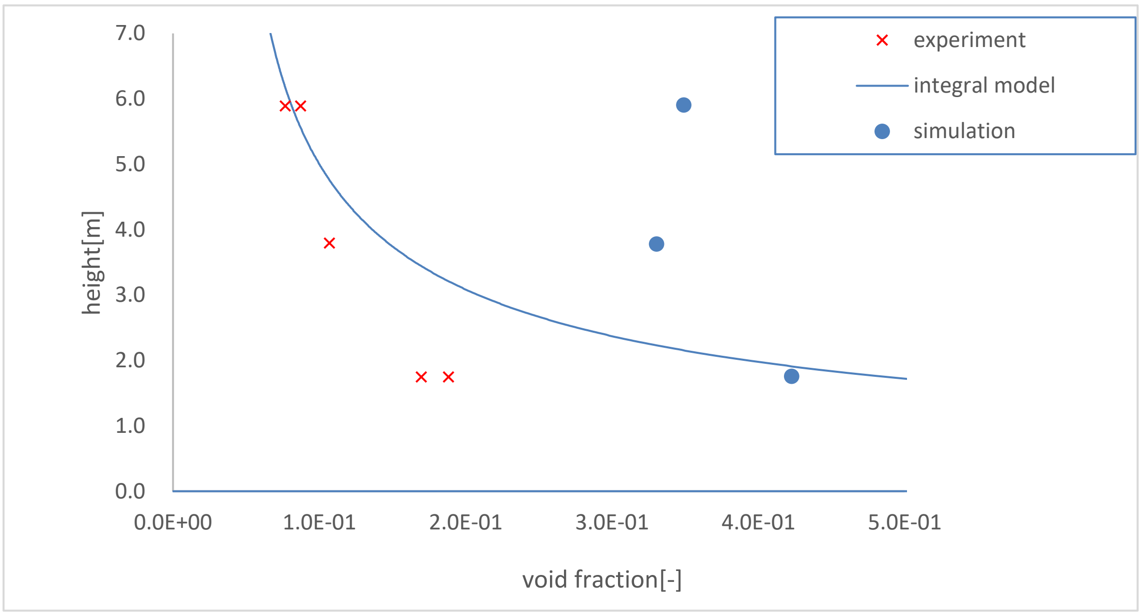

Figure 4.27: Void fraction

5 Additional validations

To test the validity of the CFD model and the integral model further, additional validations are done with the varied sizes of depth and flow rate. There are some experiments done with respect to the subsea blowout. Fanneløp's experiment(1980)[15] seems to provide more detailed experimental data, especially regarding plume width compared to Engebretsen's experiment.

Milgram's experiment(1983)[16] was done in a bigger scale of a basin. Also, it provides velocity measurement at higher flow rate than Engebretsen's experiment.

CFD model and the integral model are investigated comparing with the experimental data from both experiments above. The same set-ups are implemented as used in chapter 4, except for the size of geometry and flow rate.

5.1 Milgram's experiment

The experiment was done in a basin of the size of 50m depth at the air flow rates of 24 Nl/s, 118 Nl/s, 283 Nl/s and 590 Nl/s. Since bigger basin size and higher flow rate are of interest, only the data of the flow rate of 590 Nl/s is compared.

The integral model was tuned by determining the coefficients that show the best fits for velocity, void fraction and plume width respectively. Using γ =2 seems to show good agreement for each outcome variables. However, it is observed that each outcome is more sensitive to the value α. The α values that give the best fit are differently set for each outcome.

5.1.1 Velocity

The result of the CFD model is observed to give a good result with some deviation from the experimental data. It could be due to the dissolution of the air into water. Since the basin of Milgram's experiment is larger than the basin that was used in Engebretsen's experiment, which would have caused a longer stay time of gas in the water. it is assumed that if the model includes dissolution effect, it could give a better result.

α = 0.175 is determined in the integral model. It seems to show good agreement. As same as the CFD model, gas dissolution is not included in the integral model. The velocity is shown in Figure 5.1.

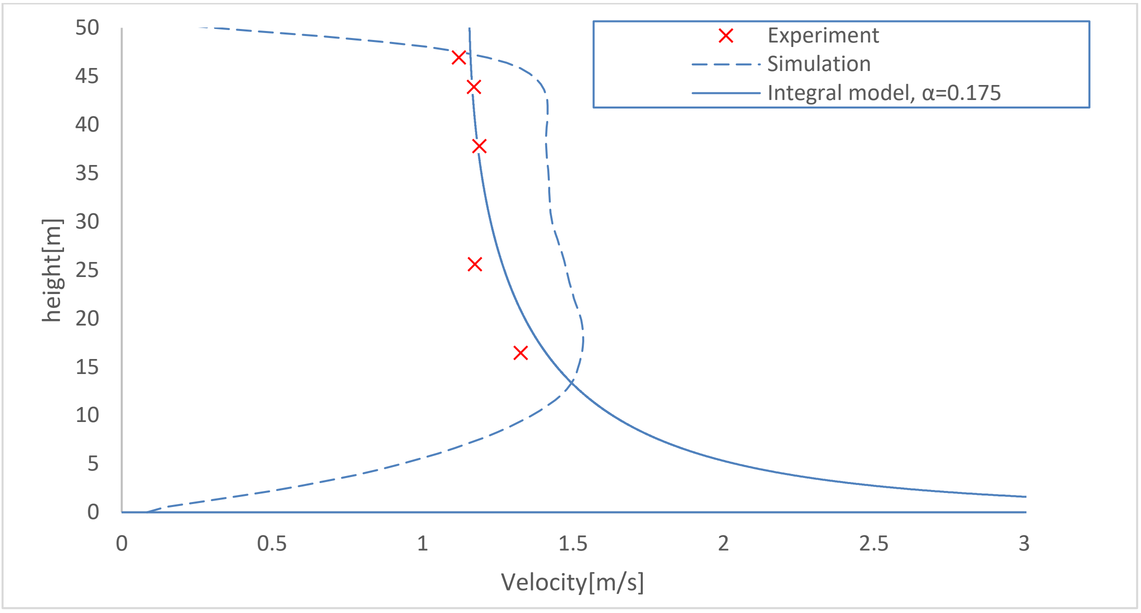

Figure 5.1: Velocity

5.1.2 Void fraction & Plume width

The CFD model give considerable deviation in void fraction. It could have been caused by gas dissolution for both models. The best fit of the void fraction from the integral model is found at $\alpha = 0.25$. The comparison of void fraction is shown in Figure 5.2.

In the CFD model, it was not clear to define the plume width. The plume width is measured by measuring the distance from the center of the plume to the point where void fraction is below 1e-02. The best fit of the plume width from the integral model is found at $\alpha = 0.13$ as shown in Figure 5.3.

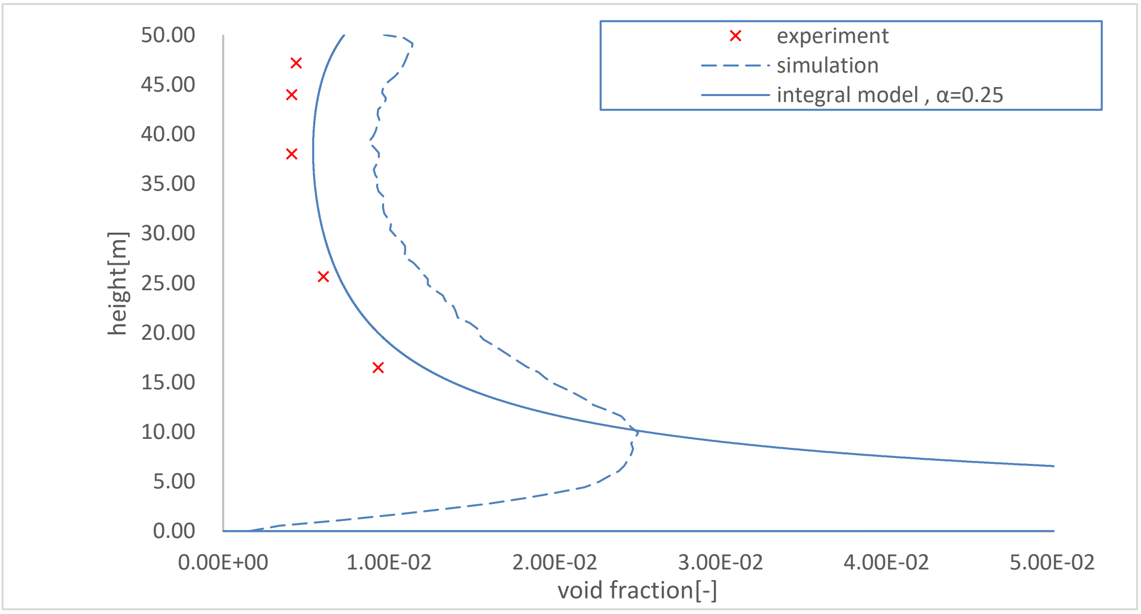

Figure 5.2: Void fraction

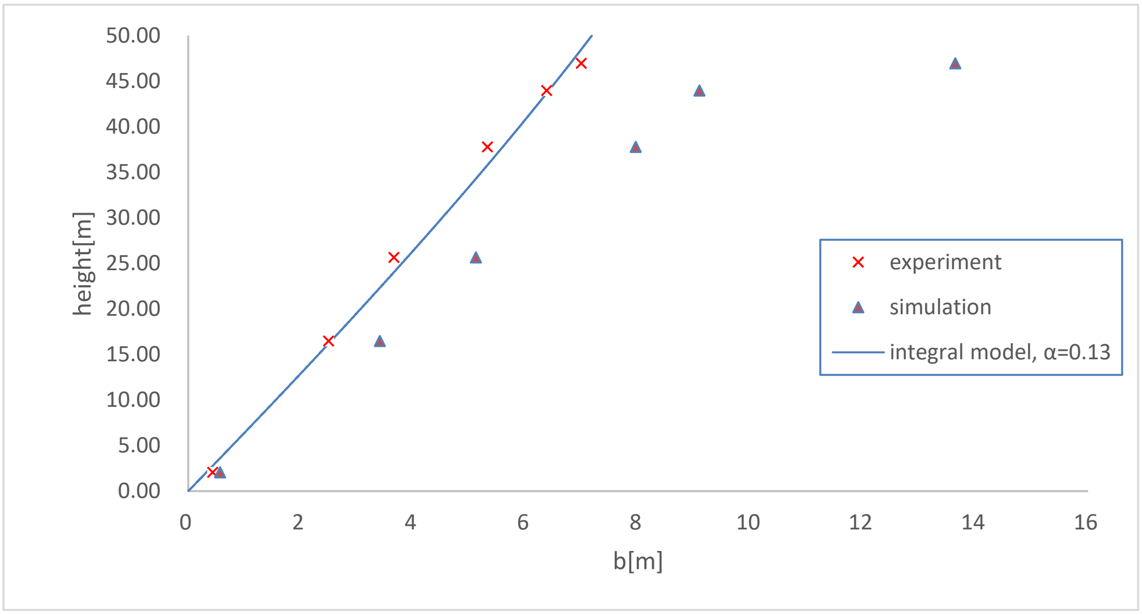

Figure 5.3: Plume width

5.2 Fanneløp's experiment

The experiment was done in a basin of the size of 10m depth at the air flow rates of 5 Nl/s, 10 Nl/s, 15 Nl/s and 22 Nl/s. Velocity profiles and void fraction profiles were obtained at 6 depths in the center of the basin. The readings of the result were averaged over 10 min during the experiment.

The CFD model is set to measure time sampled data at every second. Due to the difficulties of measuring plume width in CFD model as described in 5.1.2, it is decided not to investigate plume width.

The integral model was tuned to the coefficients of α = 0.12 and γ =1 based on the experimental data of flow rate of 5 Nl/s.

5.2.1 Velocity

The result of the CFD model is slightly underpredicted except that it shows slightly overpredicted result at the flow rate of 22Nl/s.

The integral model slightly overpredicts the velocity at every flow rate.

Velocity from both the CFD model and the integral model show good agreement. The biggest deviation from the integral model is observed at the highest flow rate of 22 Nl/s. It is due to the model tuned based on 5 Nl/s. The results are shown in Figure 5.4~5.7.

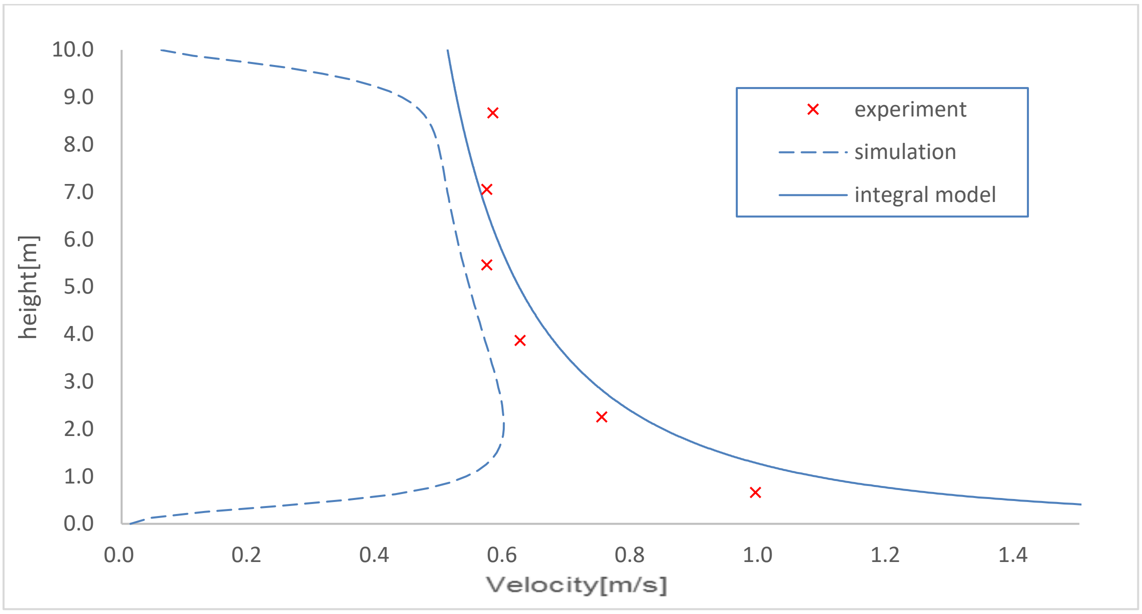

Figure 5.4: Velocity at the flow rate of 5Nl/s

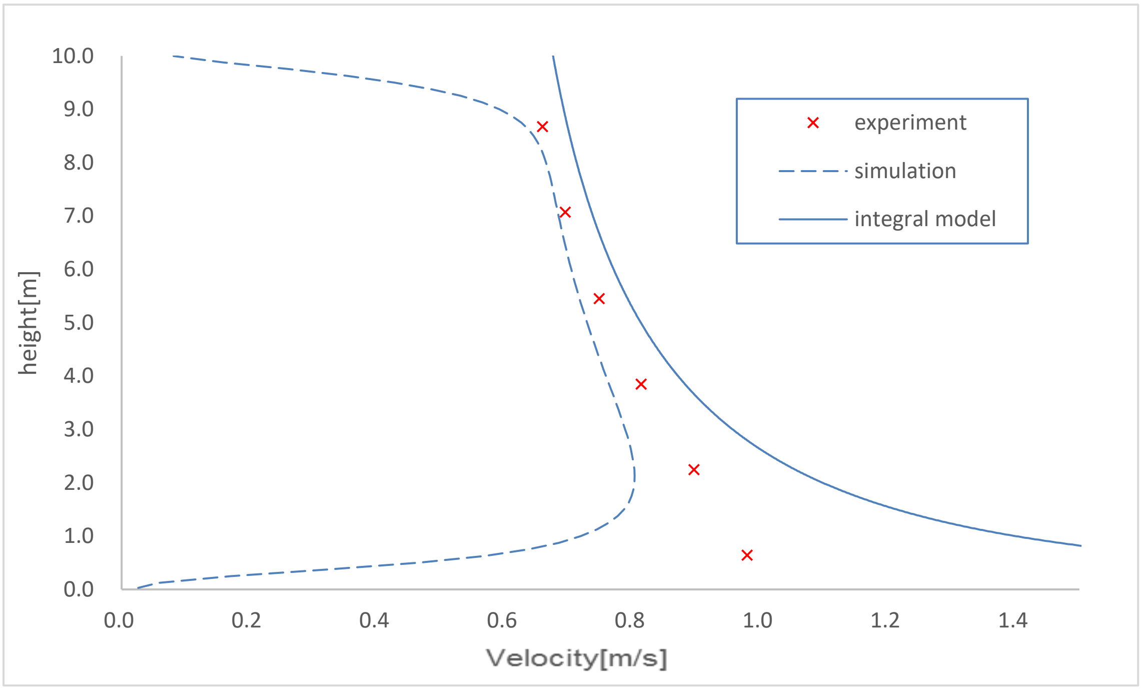

Figure 5.5: Velocity at the flow rate of 10Nl/s

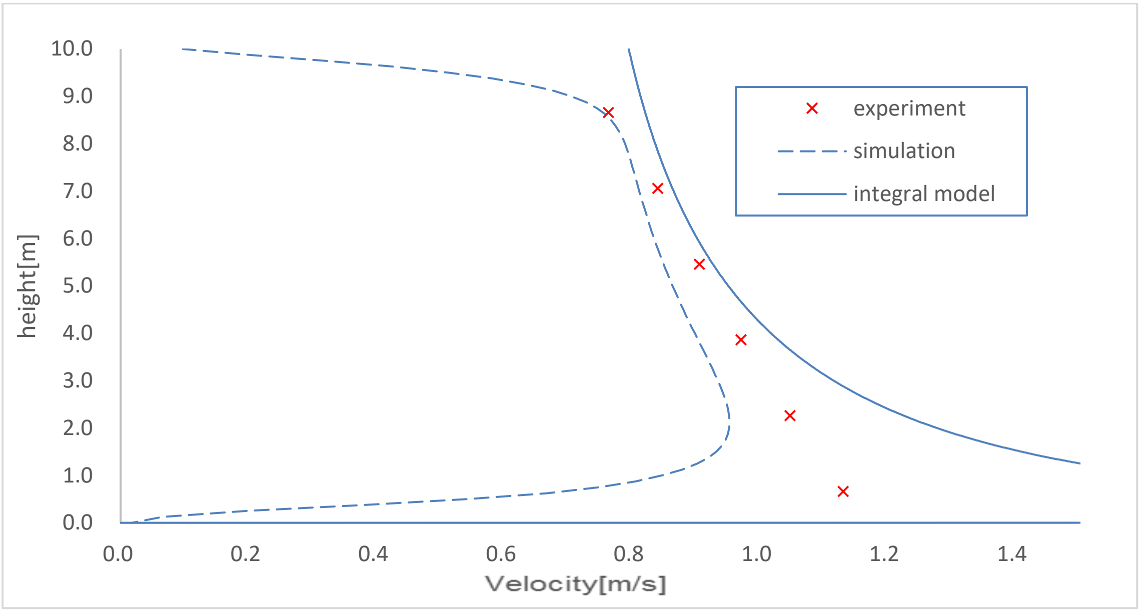

Figure 5.6: Velocity at the flow rate of 15Nl/s

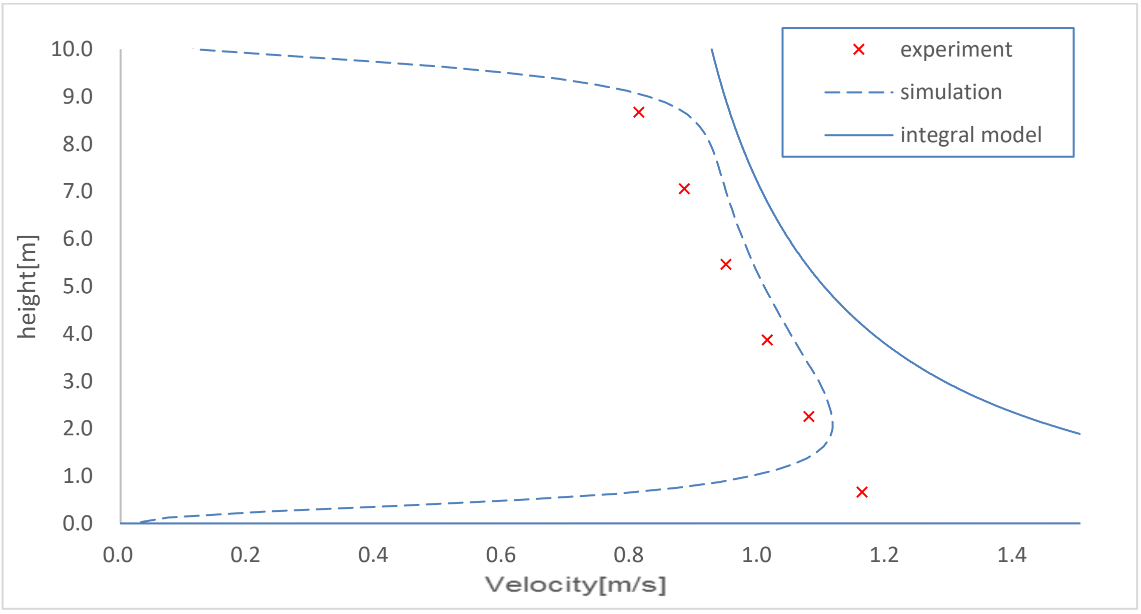

Figure 5.7: Velocity at the flow rate of 22Nl/s

5.2.2 Void fraction

While the integral model shows good agreement in void fraction at every flow rate, the CFD model result in considerable deviation at higher flow rate. The results are shown in Figure 5.8~5.11.

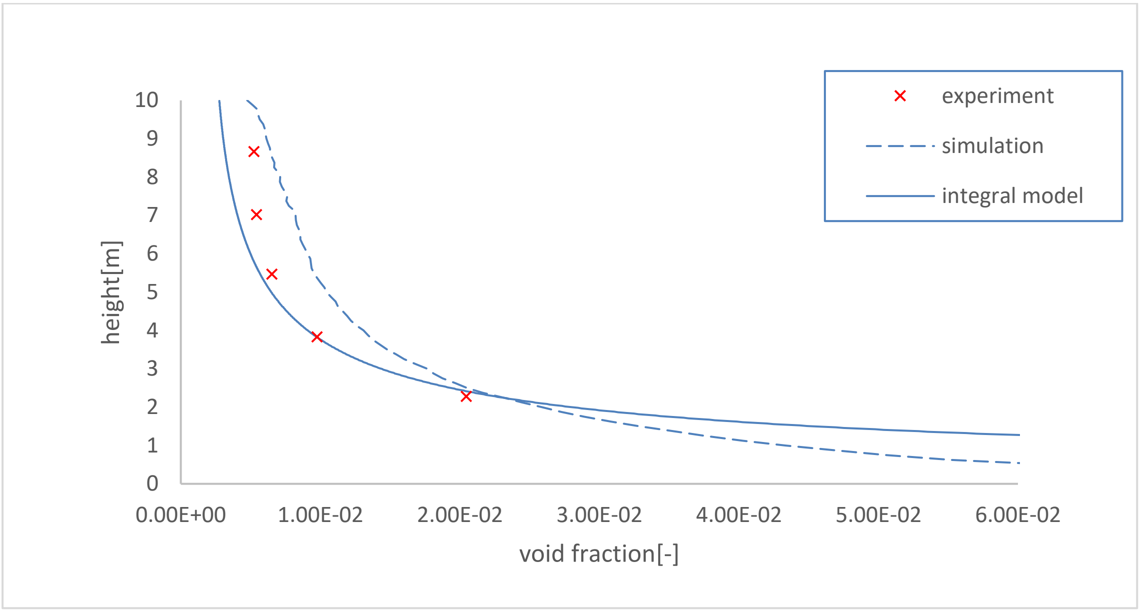

Figure 5.8: Void fraction at the flow rate of 5Nl/s

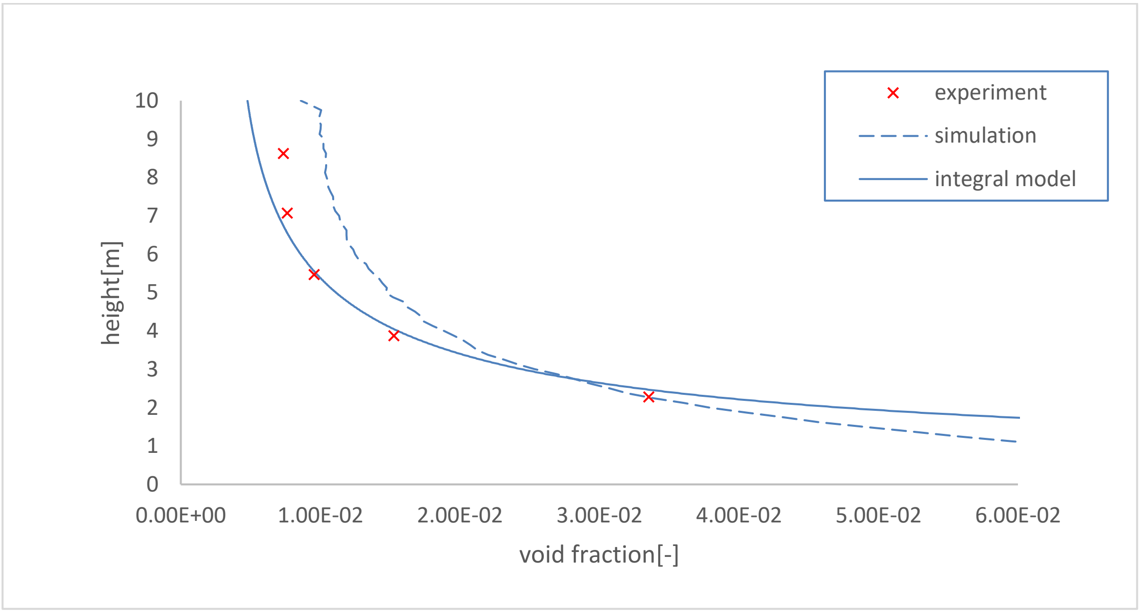

Figure 5.9: Void fraction at the flow rate of 10Nl/s

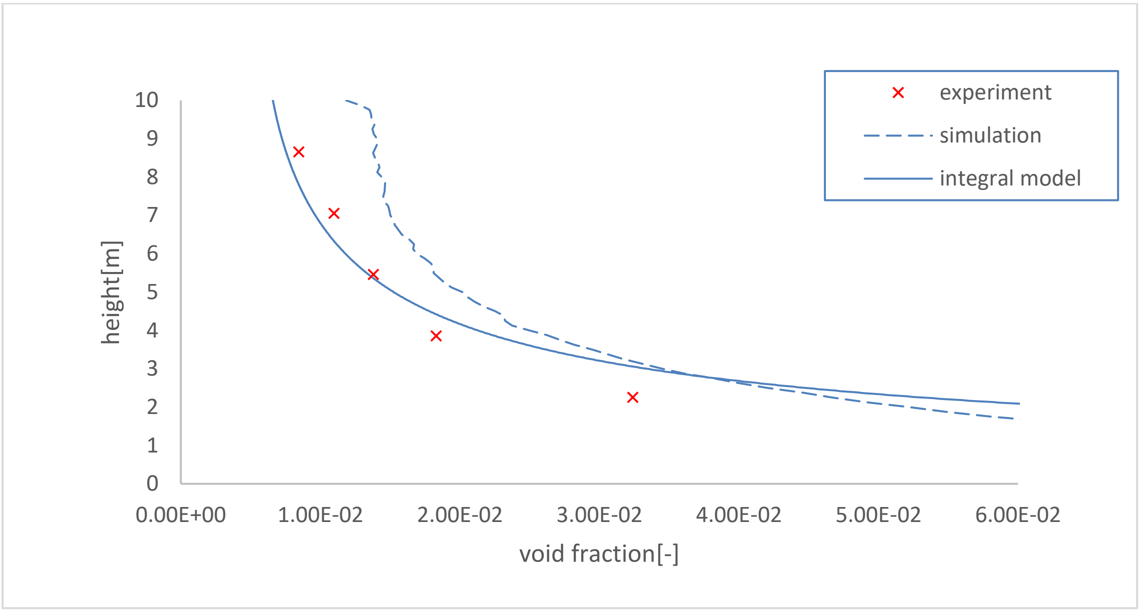

Figure 5.10: Void fraction at the flow rate of 15Nl/s

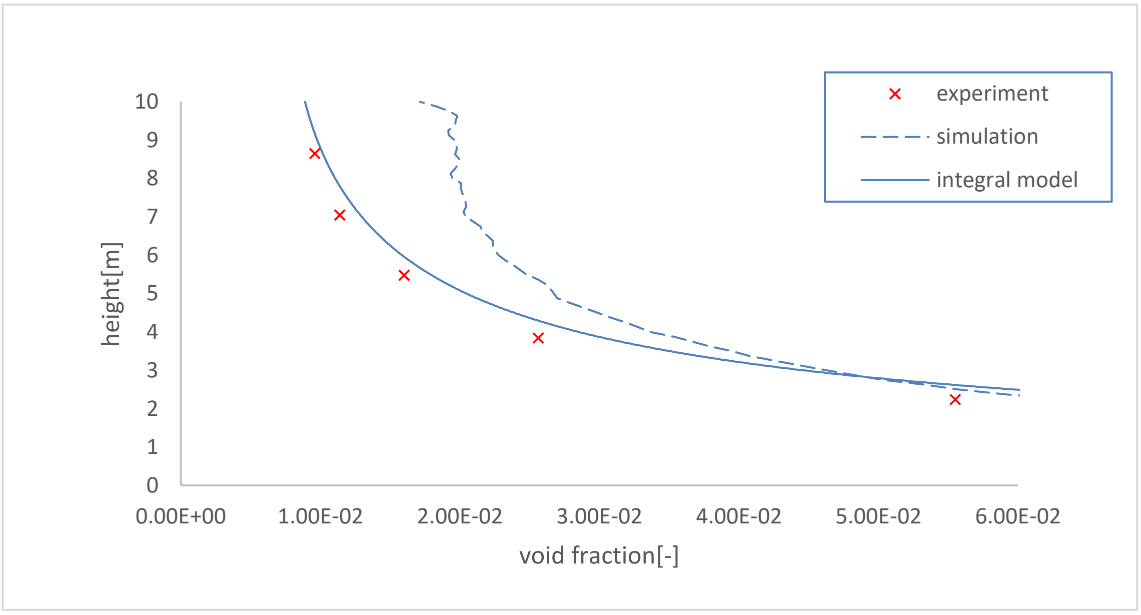

Figure 5.11: Void fraction at the flow rate of 22Nl/s

6 OpenFoam simulation

OpenFoam is a C++ based free CFD software. In this chapter, the outcome of the simulation done in OpenFoam will be presented.

To find a properly coupled solver(Eulerian-Lagrangian), different extended versions of OpenFoam were attempted. And OpenFoam v16.06+ that was developed by ESI Group contains the coupled solver.

In the following, mainly the solver will be introduced along with the result. Other set-ups such as initial conditions, constants, schemes etc. are attached in Appendix C~E.

6.1 Solver

MPPICInterFoam is the solver in which MPPICFoam and InterFoam are combined. InterFoam is the solver for VOF and MPPICFoam is the solver for DPM with a colliding particle cloud. MPPICInterFoam comprises files in a hierarchical structure as shown in Figure 6.1.

.C and .H are the extension for source file and header file respectively.

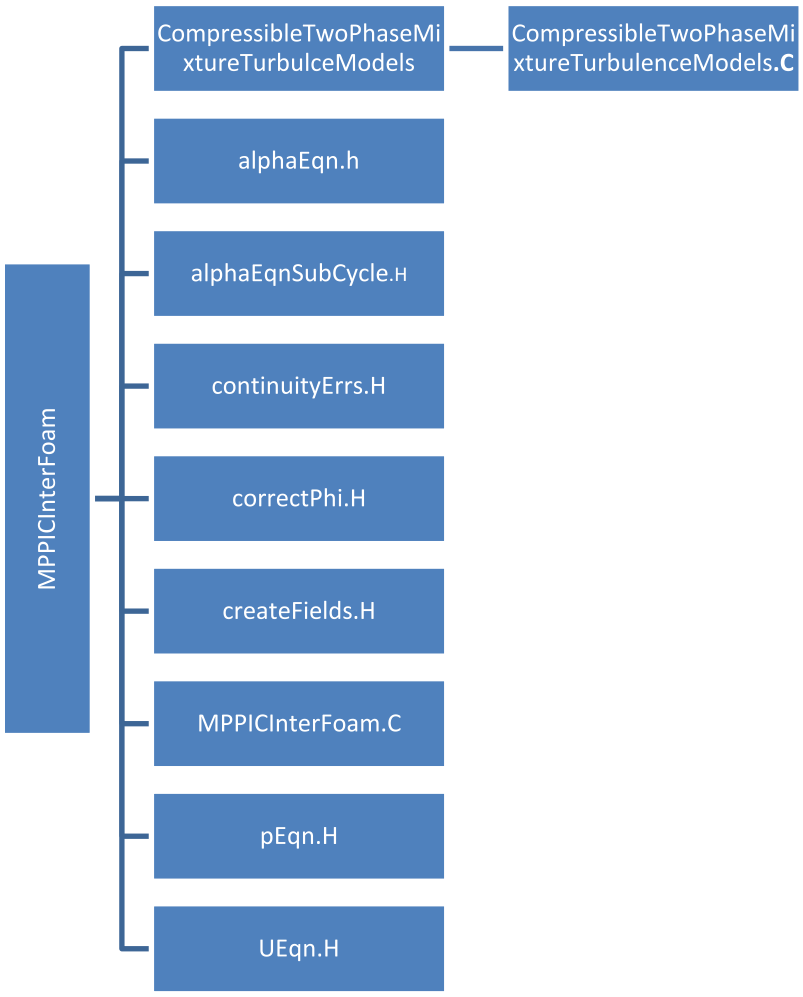

Figure 6.1: Structure of MPPICInterFoam

6.2 Result

Figure 6.2 shows a plume rising, however, the fountain of significant height is not observed during the simulation.

Figure 6.3 shows the velocity profiles.

The outcome of the simulation is not satisfying. One of the possible causes is that rectangular patch injection was implemented instead of cone injection due to the explosion of courant number during simulation. Another possible cause is the implementation of colliding particles. Lastly, when the solver was modified by me, a few functions were implemented by modifying the code and compiling the solver since the raw solver does not include necessary functions for the simulation. errors might have been caused possibly in this process. However, allowing for all the possible causes mentioned above, the velocity profiles were largely underpredicted compared to the experimental data as shown in Figure 6.3. In addition, almost invisible change in the water surface cannot be explained enough by those possible causes above. This issue seemed to be caused not by possible wrong set-ups but solver.

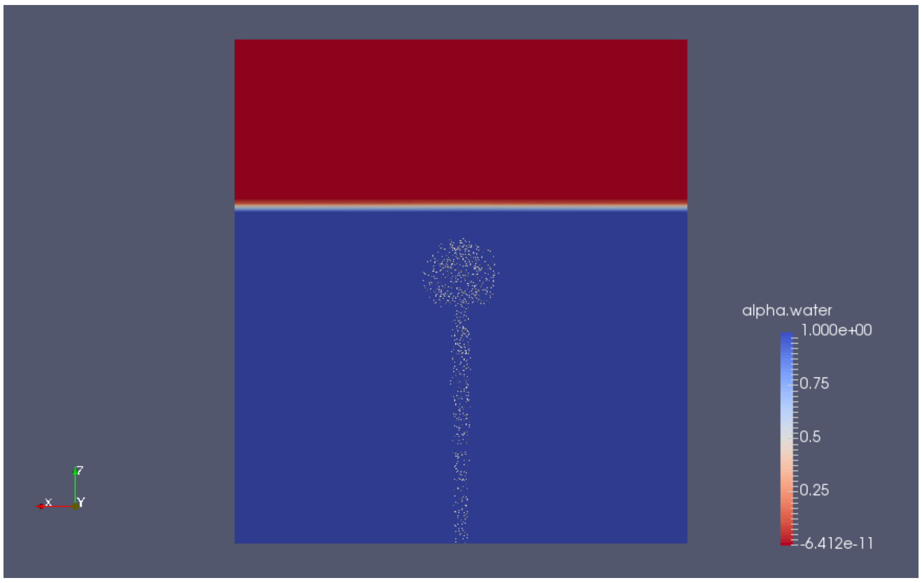

Figure 6.2: Plume

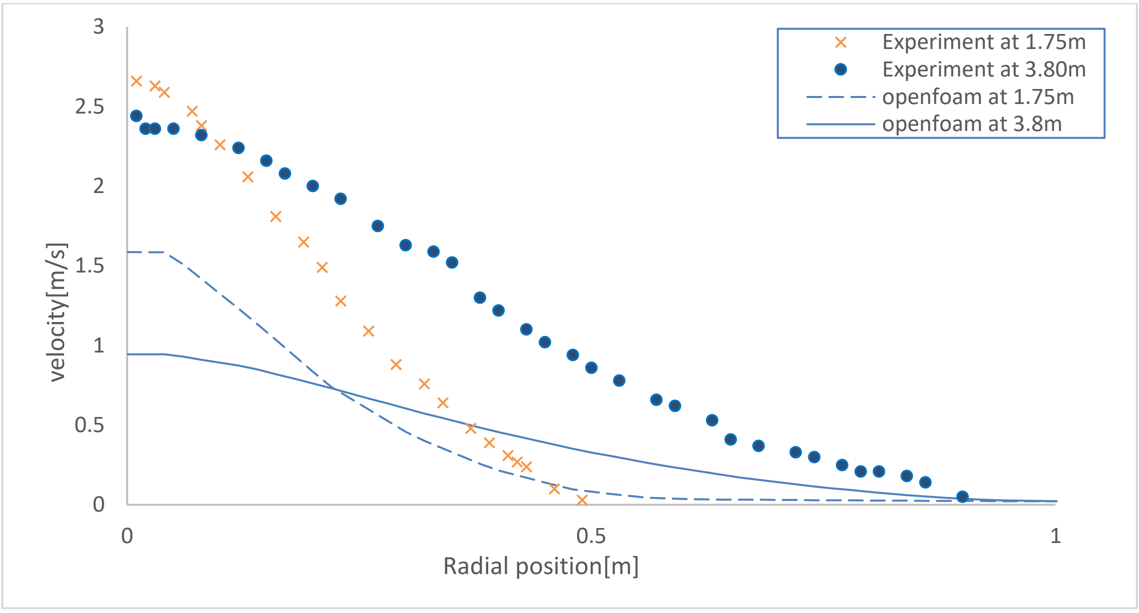

Figure 6.3: Velocity

The developer of OpenFoam 1606+, Sergio Ferraris mentioned on this issue that “The particles interact with the fluid (the phase there are in), but there is an extra small force on particles that avoids the particles to cross the interface. This force is only applied when the particle is near the interface only. Then, if the interface is affected by the particles momentum, this model will not have the desired effect.” In short, the solver is not perfectly coupled.

In the Ueqn.h file, it is found that a source term is missing transferring from particles. To fix this problem, Mainly, the source term from particles should be added in momentum at line 46 in the file as described below.

```
28 if (pimple.momentumPredictor())
29 {
30     solve
31     (
32         UEqn
33      ==
34         fvc::reconstruct
35         (
36             phicForces/rAUcf
37           +
38             (
39                 fvc::interpolate
40                 (
41                     mixture.sigmaK()
42                 )*fvc::snGrad(alpha1)
43                 - ghf*fvc::snGrad(rho)
44                 - fvc::snGrad(p_rgh)
45             ) * mesh.magSf()
46         )+Momentum from particle
47     );
48
49     fvOptions.correct(U);
50 }
```

7 Conclusion

The main object of this thesis was originally to validate the transient subsea gas plume model named ‘Rising cap model’ by comparing its outcome to CFD model and vice versa. However, Rising cap model’s main outcome is mass flux through the water surface and does not represent underwater physics. In the meantime, the CFD model which is used in this thesis was validated based on underwater velocity. In addition, the Gaussian profile assumption of surface flux in Rising cap model seems to be against the real phenomena on the surface. Therefore, it was decided not to use Rising cap model, instead Friedl’s integral model which is steady-state plume model was introduced. Friedl’s model also includes Gaussian profile, but the model accounts for the underwater physics.

The main idea of the CFD model that was proposed by Cloete is to couple Lagrangian description and Eulerian description in two-way coupling. The model includes the change of bubble’s size and density and the drag accounting for bubble’s shape. Cone injection with Random walk model is implemented to account for turbulent dispersion of bubbles.

The CFD model seems to predict underwater blowout well in terms of velocity. For more accurate prediction of the velocity, damping effect at the surface needs to be implemented [14] and dissolution of water also needs to be included depending on bubble stay time of gas. However, in terms of fountain prediction, more improvement seems to be needed.

Drag in the CFD model seems to be a factor that could affect plume physics as presented. The result by different drags shows that the drag laws including bubble’s shape show better agreement than the ones without including bubble’s shape in general.

The CFD model shows generally good agreement under different conditions such as different depth and flow rate of releasing gas. However, still to validate this model more realistically, more experimental data is necessary with higher flow rate of the gas and larger depth as well.

As a feedback of the CFD simulation done in this thesis, since it is available to measure particles void fraction when time sampling is set, maybe longer simulation should have been implemented to measure more accurate results of void fraction and plume width in steady state.

In the integral model, tuning of coefficients is necessary under different conditions. Appropriate tunings show good agreement. However, to determine the appropriate coefficients, it also needs further experiments as same as the CFD model does, with high flow rate of gas release and larger depth, which is more realistic subsea blowout condition.

Since Friedl’s integral model uses Gaussian profile as same as most of integral plume models use, the model does not account for flow’s reflection at the water surface. Modeling the area near the surface separately including reflection phenomena could be a way to improve the integral model.

Integral model is computationally cheaper and more effective in comparison to CFD model.

OpenFoam model was attempted, however, due to lack of time the model is not completed. Relatively new solver named ‘MPPICInterFoam’ is used. The solver is not exactly 2 way-coupled. It needs to be modified by inserting source term transferring from particles. In addition, the explosion of courant number while a cone injection is implemented needs to be investigated.

References

[1] G. Taylor, "The action of a surface current used as a breakwater," in *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 1955, pp. 466-478.

[2] S. Cloete, J. E. Olsen, and P. Skjetne, "CFD modeling of plume and free surface behavior resulting from a sub-sea gas release," *Applied Ocean Research,* vol. 31, pp. 220-225, 2009.

[3] T. Engebretsen, T. Northug, K. Sjøen, and T. Fanneløp, "Surface flow and gas dispersion from a subsea release of natural gas," in *The Seventh International Offshore and Polar Engineering Conference*, 1997.

[4] M. J. Friedl and T. K. Fanneløp, "Bubble plumes and their interaction with the water surface," *Applied Ocean Research,* vol. 22, pp. 119-128, 4// 2000.

[5] A. Fluent, "Theory Guide and User's Guide," *Ansys Inc, USA*, 2015.

[6] R. Clift, J. R. Grace, and M. E. Weber, *Bubbles, drops, and particles*: Courier Corporation, 2005.

[7] J. Xia, T. Ahokainen, and L. Holappa, "Analysis of flows in a ladle with gas-stirred melt," *Scandinavian journal of metallurgy,* vol. 30, pp. 69-76, 2001.

[8] T. Z. Harmathy, "Velocity of large drops and bubbles in media of infinite or restricted extent," *AIChE Journal,* vol. 6, pp. 281-288, 1960.

[9] M. Bakli, "Evaluation of Gas and Oil Dispersion during Subsea Blowouts," Institutt for energi-og prosessteknikk, 2014.

[10] S. V. Patankar and D. B. Spalding, "A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows," *International journal of heat and mass transfer,* vol. 15, pp. 1787-1806, 1972.

[11] H. K. Versteeg and W. Malalasekera, *An introduction to computational fluid dynamics: the finite volume method*: Pearson Education, 2007.

[12] R. I. Issa, "Solution of the implicitly discretised fluid flow equations by operator-splitting," *J. Comput. Phys.,* vol. 62, pp. 40-65, 1986.

[13] H. Laux and S. T. Johansen, "A CFD analysis of the air entrainment rate due to a plunging steel jet combining mathematical models for dispersed and separated multiphase flows," *Fluid Flow Phenomena in Metal Processing,* 1999.

[14] Q. Q. Pan, J. E. Olsen, S. T. Johansen, M. Reed, and L. R. Sætran, "CFD Study of Surface Flow and Gas Dispersion From a Subsea Gas Release," in *ASME 2014 33rd International Conference on Ocean, Offshore and Arctic Engineering*, 2014, pp. V007T12A032-V007T12A032.

[15] T. Fannelop and K. Sjoen, "Hydrodynamics of underwater blowouts," in *18th Aerospace Sciences Meeting*, 1980, p. 219.

[16] J. Milgram, "Mean flow in round bubble plumes," *Journal of Fluid Mechanics,* vol. 133, pp. 345-376, 1983.

HSN University College of Southeast Norway

Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

FMH606 Master's Thesis

Title: CFD Validation of transient subsea gas plume model

TUC supervisor: Associate Professor Amaranath S. Kumara

External partner: Lloyd's Register Consulting

Task background:
Subsea gas releases are often a significant risk contributor to offshore installations. Hazards such as dropped objects, fatigue, corrosion etc. may results in gas leakages from subsea pipelines or installations. A subsea gas leakage and the subsequent gas dispersion is a complex physical phenomenon which typically includes sonic flow, multiphase flow, strong turbulence and very high Reynolds numbers. The physical process will inevitable be subject to rather rough modelling when the topside risk is to be estimated.

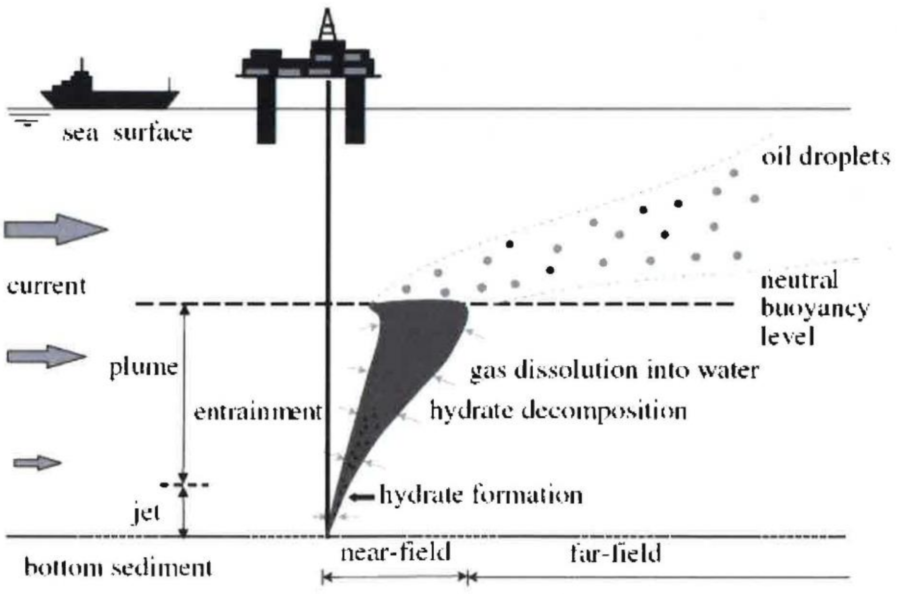

Figure 1: Scheme of deep water oil/gas spill

The Lloyd's Register rising cap model (in-house analytical model) estimates the surface release of gas from a subsea gas leakage. Based on the leakage flow rate, the depth and the gas density, the model will predict the progress of the gas rising towards the surface. The model assumes that the gas leakage will form a plume, and subsequently the plume will form a cap above itself. The cap travels upward, gaining mass and momentum from the plume underneath. In addition, the surrounding water will entrain the cap and contribute to its growth. When the cap reaches the surface, gas is emitted to the air.

Address: Kjølnes ring 56, NO-3918 Porsgrunn, Norway. **Phone:** 35 57 50 00. **Fax:** 35 55 75 47.

Task description:
The main objective of this project is to validate transient subsea gas plume model using CFD software ANSYS FLUENT. The candidate will be given an opportunity to work with industrial experts in Lloyd's Register Consulting. The project includes the following key activities:

1) Perform in-depth literature study on transient subsea gas releases and collect existing modelling and experimental data
2) Implementation of subsea gas/oil plume model on FLUENT/OpenFOAM. The model will incorporate the presence of sea currents. The density changes in the bubbles, their size variation, and gas dissolution will also be included.
3) The work can be started with a gas plume for which some experimental data and validated integral models or CFD models that can be used for comparison. A simple model for the gas bubble size that takes into account for bubble expansion should be evaluated.
4) In the case of combined gas and oil plumes, the difference in boundary conditions at sea surface must be handled. The gas bubbles should be allowed to escape while oil bubbles must stay in the water column. Validate the model against existing experimental data and models.

Student category: PT, EET, SCE

Practical arrangements:
All the practical issues will be discussed at the kick-off meeting.
Software: Lloyd's Register Consulting will have exclusive rights for software for three years.

Signatures:

Student (date and signature):

02/03/2017 [illegible]

Supervisor (date and signature):

23/02/2017 [illegible]

Appendices

Appendix A-1 < Spherical drag >

Range of Re	a_1	a_2	a_3
0<Re<0.1	0	24	0
0.1<Re<1	3.690	22.73	0.0903
1<Re<10	1.222	29.1667	-3.8889
10<Re<100	0.6167	46.50	-116.67
100<Re<1000	0.3644	98.33	-2778
1000<Re<5000	0.357	148.62	-47500
5000<Re<10000	0.46	-490.546	578700
$10000 \leq \text{Re}$	0.5191	-1662.5	5416700

Appendix A-2 <Modified sphere drag >

Range of Re	Correlation
Re<0.1	$C_D = \frac{3}{16} + \frac{24}{\text{Re}}$
$0.01 < \text{Re} \leq 20$	$C_D = \frac{24}{\text{Re}}\left[1 + 0.1315\text{Re}^{(0.82-0.05w)}\right]$
$20 \leq \text{Re} \leq 260$	$C_D = \frac{24}{\text{Re}}\left[1 + 0.1935\text{Re}^{0.6305}\right]$
$260 \leq \text{Re} \leq 1500$	$C_D = 10^{1.6435-1.1242w+0.1558w^2}$
$1.5\times10^3 \leq \text{Re} \leq 1.2\times10^4$	$C_D = 10^{-2.4571+2.5558w-0.9295w^2+0.1049w^3}$
$1.2\times10^4 \leq \text{Re} \leq 4.4\times10^4$	$C_D = 10^{-1.9181+0.6370w-0.0636w^2}$
$4.4\times10^4 \leq \text{Re} \leq 3.38\times10^5$	$C_D = 10^{-4.3390+1.5809w-0.1546w^2}$
$3.38\times10^5 \leq \text{Re} \leq 4\times10^5$	$C_D = 29.78 - 5.3w$
$4\times10^5 \leq \text{Re} \leq 10^6$	$C_D = 0.1w - 0.49$

$10^6 \le \mathrm{Re}$	$C_D = 0.19 - \frac{8 \times 10^4}{\mathrm{Re}}$

($w = \log_{10} \mathrm{Re}$)

Appendix B-1

```
%% Integral plume model-MJ Fridedl
% Author: Kim Taewook
% Linkedin : https://www.linkedin.com/in/taewook-kim/
% GITHUB : github.com/Kimtaewookcode
% Email : kimtaewook87@gmail.com
% Based on
% http://www.sciencedirect.com/science/article/pii/S014111879900022X
clear
close all
clc
g = 9.81;%gravity[m/s2]
rho_w=998;%density of water[kg/s]
m_release=0.02695;%0.01838;%0.00625;%0.71milgram%0.208;%mass rate[kg/s]
phi=3.14;%phi
p0=101325;%atmopheric pressure[pa]
rho_g0=1.225;%density of releasing gas[kg/m3]
vs=0.35;%slip velocity[m/s]
h=10;%height of the water surface
hp=10.33;
x1 = 0;%centerline
hoff=0;%h offset
z0=1.75;%z value where you want to know about profile at
%coefficients%
alpha=0.12;%0.13;%0.1285;entrainment coefficient
gam=1;%1.5;
beta=0.39;%0.5;theoretical value%0.39;experimental value : for a loss-free
rise;1for instantneous
lambda=0.8;

%%calculation in x(radial) direction%%
vs1=vs*((g*m_release*(lambda^2+1))/((h+hp)*phi*gam*rho_g0*2*alpha^2))^(-
1/3);
s1=(1+lambda^2)*vs1;
z1=z0/(h+hp);
z2=h/(h+hp);
v1=((25/12)^(1/3))*(z1^(-1/3))*(1+11*z1/39+511/2*(z1/39)^2)-
s1*7/22*(1+345/343*z1/13+86175/11662*(z1/13)^2)+(s1^2)*13/121*(12/25)^(1/3)
*(z1^(1/2))*(1-59489/1436*z1/39-2825583625/23347324*(z1/39)^2);
v2=((25/12)^(1/3))*(z2^(-1/3))*(1+11*z2/39+511/2*(z2/39)^2)-
s1*7/22*(1+345/343*z2/13+86175/11662*(z2/13)^2)+(s1^2)*13/121*(12/25)^(1/3)
*(z2^(1/2))*(1-59489/1436*z2/39-2825583625/23347324*(z2/39)^2);
v=v1*((g*m_release*(lambda^2+1))/((h+hp)*phi*gam*rho_g0*2*alpha^2))^(1/3);
vh=v2*((g*m_release*(lambda^2+1))/((h+hp)*phi*gam*rho_g0*2*alpha^2))^(1/3);
b1=3/5*z1*(1-z1/13-7*(z1/13)^2)+s1*3/110*((12/25)^(1/3))*(z1^(4/3))*(1-
1046/49*z1/39-227726/833*(z1/39)^2)-
(s1^2)*48/15121*((25/12)^(1/3))*(z1^(5/3))*(1-
34663/9408*z1+225707803/240143904*z1^2);
b2=3/5*z2*(1-z2/13-7*(z2/13)^2)+s1*3/110*((12/25)^(1/3))*(z2^(4/3))*(1-
1046/49*z2/39-227726/833*(z2/39)^2)-
(s1^2)*48/15121*((25/12)^(1/3))*(z2^(5/3))*(1-
34663/9408*z2+225707803/240143904*z2^2);
b=b1*(2*alpha*(h+hp));%plume width at z0
bh=b2*(2*alpha*(h+hp));%plume width at the surface
x = [0 : 0.01: 200];%x range- need to be set
y=v*exp(-(x.^2)/(b^2));%velocity at z0
yh=vh*exp(-(x1^2)/(b^2));%velocity at the surface

hf=beta*gam*(yh.^2)/g;%the peak of fountain profile
```

```
hr=hf*exp(-(x.^2)/(bh^2))-hoff;%fountain profile

void_1=1/(1-z1)/((b1^2)*(v1+s1));
void1=void_1*((((1+lambda^2)^2)*gam*(m_release/rho_g0)^2)/((phi^2)*(lambda^
2)*(2^5)*(alpha^4)*((h+hp)^5)*g))^(1/3);
voidr=void1*exp(-(x.^2)/((lambda^2)*(b.^2)));%void fraction profile at z0

void_2=1/(1-z2)/((b2^2)*(v2+s1));
void2=void_2*((((1+lambda^2)^2)*gam*(m_release/rho_g0)^2)/((phi^2)*(lambda^
2)*(2^5)*(alpha^4)*((h+hp)^5)*g))^(1/3);
voidh=void2*exp(-(x.^2)/((lambda^2)*(b^2)));%void fraction profile at the
surface
mflux_surf=voidh*yh*rho_g0;%mass flux per unit area at the surface

plot(x, y, '.-')
xlim([0 4])
title(['velocity at z=',num2str(z0)]);
xlabel('radial position[m]');
ylabel('velocity[m/s]');

figure;plot(x, hr, '.-')
xlim([0 4])
title('fountain ');
xlabel('radial position[m]');
ylabel('z[m]');

figure;plot(x, voidr, '.-')
xlim([0 4])
title(['void fraction at z=',num2str(z0)]);
xlabel('radial position[m]');
ylabel('voidfraction');

figure;plot(x, mflux_surf, '.-')
xlim([0 4])
title(['mass flux at the surface']);
xlabel('radial position[m]');
ylabel('massflux[kg/s/m2]');

%%%calculation in z direction%%
initial=0;%z start
last=h;%z end
dz=0.01;%delta z
n=(last-initial)/dz;%the number of data
zarr=zeros(n,1);% n x 1 array for z
i=1;

for z=initial:dz:last
zarr(i)=z;
z1=z/(h+hp);
vs1=vs*((g*m_release*(lambda^2+1))/((h+hp)*phi*gam*rho_g0*2*alpha^2))^(-
1/3);
s1=(1+lambda^2)*vs1;
v1_z=((25/12)^(1/3))*(z1.^(-1/3))*(1+11*z1/39+511/2*(z1/39).^2)-
s1*7/22*(1+345/343*z1/13+86175/11662*(z1/13).^2)+(s1^2)*13/121*(12/25)^(1/3
)*(z1.^(1/2))*(1-59489/1436*z1/39-2825583625/23347324*(z1/39).^2);
v_z=v1_z*((g*m_release*(lambda^2+1))/((h+hp)*phi*gam*rho_g0*2*alpha^2))^(1/
3);
b1_z=3/5*z1*(1-z1/13-7*(z1/13)^2)+s1*3/110*((12/25)^(1/3))*(z1^(4/3))*(1-
1046/49*z1/39-227726/833*(z1/39)^2)-
(s1^2)*48/15121*((25/12)^(1/3))*(z1^(5/3))*(1-
34663/9408*z1+225707803/240143904*z1^2);
```

```
b_z=b1_z*(2*alpha*(h+hp));
bz(i,1)=b_z;%designate br's 'i' th value
veloarr_z(i,1)=v_z*exp(-(x1^2)/(b_z^2));%designate veloarr's 'i' th value

void1_z=1/(1-z1)/((b1_z.^2)*(v1_z+s1));
void_z=void1_z*((((1+lambda^2)^2)*gam*(m_release/rho_g0)^2)/((phi^2)*(lambd
a^2)*(2^5)*(alpha^4)*((h+hp)^5)*g))^(1/3);
voidz(i,1)=void_z*exp(-(x1^2)/((lambda^2)*(b_z^2)));%designate voidr's 'i'
th value


 i=i+1;
end
figure;plot(bz,zarr, '.-')
ylim([0 h])
title('plume width');
ylabel('z position[m]');
xlabel('width[m]');
figure;plot(veloarr_z,zarr,'.-')
xlim([0 4])
ylim([0 h])
title('velocity at the centerline ');
ylabel('z position[m]');
xlabel('velocity[m/s]');
figure;plot(voidz,zarr, '.-')
xlim([0 0.06])
ylim([0 h])
title('voidfraction at the centerline');
ylabel('z position[m]');
xlabel('voidfraction');
instantneous_maximum_fountain=3.1*hf;
instantneous_average_fountain=2*hf;
```

Appendix B-2

```
%Central differencing scheme,1-D convection-diffusion steady state, no
source but only boundary
%condition
%Author : Taewook Kim
%LINKEDIN : www.linkedin.com/in/kimtw
%GITHUB : github.com/Kimtaewookcode
%Email : kimtaewook87@gmail.com
%Ref:"An introduction to Computational Fluid Dynamics:HKversteeg and
%WMalalasekera example 5.1
%%%%%%%%
clear
clc
clf
L=1;%[m] length of geometry
%rho=1;%[kg/m3]
ksi=0.1;%[kg/ms]
u=2.5;%[m/s]%2.5;
ngrid=20;%numberof grid%20,5
dx=L/ngrid;%delta x

F=2.5;%coefficient
D=ksi/dx;%coefficient

phi0=1;%boundary condition at x=0
phil=0;%boundary condition at x=L

%%%%%%%%
array=zeros(ngrid);
bound=zeros(ngrid,1);
%%%%%%%%%

for i=1:ngrid
    if i==1
        aw=0;
        ae=D-F/2;
        sp=-(2*D+F);
        ap=aw+ae-sp;
        su=-sp*phi0;
        array(i,i)=ap;
        array(i,i+1)=-ae;
        bound(i,1)=su;
    elseif i<ngrid
        aw=D+F/2;
        ae=D-F/2;
        sp=0;
        ap=aw+ae-sp;
        su=0;%no source
        array(i,i-1)=-aw;
        array(i,i)=ap;
        array(i,i+1)=-ae;
        bound(i,1)=su;

    else
        aw=D+F/2;
        ae=0;
        sp=-(2*D-F);
        ap=aw+ae-sp;
        su=-sp*phil;

        array(i,i-1)=-aw;
```

```
         array(i,i)=ap;
         bound(i,1)=su;
     end
 end
  phi=array\bound;

  xarr=zeros(ngrid,1);
  for j=1:ngrid
      xarr(j,1)=dx/2+dx*(j-1);
  end

  %%%%%%%analytical
  dx1=dx/10;
  x1=[0:dx1:L];
  anaphi=1+(1-exp(25*x1))/(7.20*10^10);

  plot(xarr,phi,'x',x1,anaphi,'-')
  ylim([0 1.5])
```

Appendix B-3

```
%Upwind scheme,1-D convection-diffusion steady state, no source but only
boundary
%condition
%Author : Taewook Kim
%LINKEDIN : www.linkedin.com/in/kimtw
%GITHUB : github.com/Kimtaewookcode
%Email  : kimtaewook87@gmail.com
%Ref:"An introduction to Computational Fluid Dynamics:HKversteeg and
%WMalalasekera example 5.2
%%%%%%%%
clear
clc
clf
L=1;%[m] length of geometry
%rho=1;%[kg/m3]
ksi=0.1;%[kg/ms]
u=2.5;%[m/s]%2.5;
ngrid=20;%numberof grid%20,5
dx=L/ngrid;%delta x

F=2.5;%coefficient
D=ksi/dx;%coefficient

phi0=1;%boundary condition at x=0
phil=0;%boundary condition at x=L

%%%%%%%%
array=zeros(ngrid);
bound=zeros(ngrid,1);
%%%%%%%%%

for i=1:ngrid
    if i==1
        aw=0;
        ae=D;
        sp=-(2*D+F);
        ap=aw+ae-sp;
        su=-sp*phi0;
        array(i,i)=ap;
        array(i,i+1)=-ae;
        bound(i,1)=su;
    elseif i<ngrid
        aw=D+F;
        ae=D;
        sp=0;
        ap=aw+ae-sp;
        su=0;%no source
        array(i,i-1)=-aw;
        array(i,i)=ap;
        array(i,i+1)=-ae;
        bound(i,1)=su;

    else
        aw=D+F;
        ae=0;
        sp=-(2*D);
        ap=aw+ae-sp;
        su=-sp*phil;

        array(i,i-1)=-aw;
```

```
       array(i,i)=ap;
       bound(i,1)=su;
   end
end
 phi=array\bound;

 xarr=zeros(ngrid,1);
 for j=1:ngrid
     xarr(j,1)=dx/2+dx*(j-1);
 end

 %%%%%%%analytical
 dx1=dx/10;
 x1=[0:dx1:L];
 anaphi=1+(1-exp(25*x1))/(7.20*10^10);

 plot(xarr,phi,'x',x1,anaphi,'-')
```

Appendix B-4

```
%Second order upwind scheme,1-D convection-diffusion steady state, no
source but only boundary
%condition
%Author : Taewook Kim
%LINKEDIN : www.linkedin.com/in/kimtw
%GITHUB : github.com/Kimtaewookcode
%Email : kimtaewook87@gmail.com
%Ref:"Lars Davidson: Numerical Methods for Turbulent Flow chapter5
%%%%%%%%
clear
clc
clf
L=1;%[m] length of geometry
%rho=1;%[kg/m3]
ksi=0.1;%[kg/ms]
u=2.5;%[m/s]%2.5;
ngrid=20;%numberof grid%20,5
dx=L/ngrid;%delta x

F=2.5;%coefficient
D=ksi/dx;%coefficient

phi0=1;%boundary condition at x=0
phil=0;%boundary condition at x=L

%%%%%%%%
array=zeros(ngrid);
bound=zeros(ngrid,1);
%%%%%%%%%

for i=1:ngrid
    if i==1
        aw=0;
        ae=D;
        ap=3/2*F+3*D;
        su=(3/2*F+2*D)*phi0;
        array(i,i)=ap;
        array(i,i+1)=-ae;
        bound(i,1)=su;
    elseif i==2
        aw=D+2*F;
        ae=D;
        ap=3/2*F+2*D;
        su=-F/2*phi0;%no source
        array(i,i-1)=-aw;
        array(i,i)=ap;
        array(i,i+1)=-ae;
        bound(i,1)=su;

    elseif i<ngrid
        aw=D+2*F;
        ae=D;
        ap=3/2*F+2*D;
        aww=-F/2;
        su=0;%no source
        array(i,i-2)=-aww;
        array(i,i-1)=-aw;
        array(i,i)=ap;
        array(i,i+1)=-ae;
        bound(i,1)=su;
        else
        aw=D+2*F;
        ae=0;
        ap=3/2*F+3*D;
        aww=-F/2;
        su=2*D*phil;
        array(i,i-2)=-aww;
        array(i,i-1)=-aw;
        array(i,i)=ap;
        bound(i,1)=su;
    end
end
 phi=array\bound;

 xarr=zeros(ngrid,1);
 for j=1:ngrid
     xarr(j,1)=dx/2+dx*(j-1);
 end

 %%%%%%%analytical
 dx1=dx/10;
 x1=[0:dx1:L];
 anaphi=1+(1-exp(25*x1))/(7.20*10^10);

 plot(xarr,phi,'x',x1,anaphi,'-')
```

Appendix B-5

```
%SIMPLE algorithm,2-D nozzle steady state,
%Author : Taewook Kim
%LINKEDIN : www.linkedin.com/in/kimtw
%GITHUB : github.com/Kimtaewookcode
%Email : kimtaewook87@gmail.com
%Ref:"An introduction to Computational Fluid Dynamics:HKversteeg and
%WMalalasekera example 6.2
%%%%%%%%
clear all
clc
clf

rho=1;%density
L=2;%length of geometry
nnode=5;%numberof node
nnodev=nnode-1;%number of cell
dx=L/nnodev;%delta x, uniform
AA=0.5;%Area at A
AE=0.1;%Area at E
iteration=100;%numberof maximum iteration
tor=10e-05;%maximum sum of residual
und=0.8;%underrelaxation factor
xdarrp=zeros(1,nnode);%matrix of distance for pressure points
for i=1:nnode
xdarrp(1,i)=-dx+dx*i;%matrix of x distances for pressure points
end
Axp=zeros(1,nnode);%Areas of xdarrp
Axp=0.5-(AA-AE)/L*xdarrp;

xdarrv=zeros(1,nnodev);%matrix of x distances for velocity points
for i=1:nnodev
xdarrv(1,i)=xdarrp(1,i)+dx/2;
end
Axv=0.5-(AA-AE)/L*xdarrv;%matrix of areas of velocity points

p0=10;%pressure at the inlet[pa]
pE=0;%pressure at the outlet
mdot=1;%[kg/s]
u_ini=mdot/rho./Axv;
%%%psuedo initial velocity
u_ini_p=mdot/rho./Axp;
p_ini=p0-p0/L*xdarrp;%%%p guess-linear guess
d=zeros(1,nnodev);%parameter that is used for pressure corrector
%%%%%
array=zeros(nnodev);
bound=zeros(nnodev,1);
residuals=zeros(1,nnodev);
%%%%%
for j=1:iteration

for i=1:nnodev
    if i==1
        Fw=rho*u_ini_p(1,i)*Axp(1,i);
        Fe=rho*(u_ini(1,i)+u_ini(1,i+1))/2*Axp(1,i+1);
        aW=0;
        aE=0;
        aP=Fe+Fw*0.5*(Axv(1,i)/Axp(1,i))^2;
        Su=(p0-p_ini(1,i+1))*Axv(1,i)+Fw*Axv(1,i)/Axp(1,i)*u_ini(1,i);%u_ini
is the velocity at previous iteration
        d(1,i)=Axv(1,i)/aP;
        array(i,i)=aP;
        array(i,i+1)=-aE;
        bound(i,1)=Su;
        residuals(1,i)=abs(aP*u_ini(1,i)-Su);

    elseif i<nnodev
        Fw=rho*(u_ini(1,i-1)+u_ini(1,i))/2*Axp(1,i);
        Fe=rho*(u_ini(1,i)+u_ini(1,i+1))/2*Axp(1,i+1);
        aW=Fw;
        aE=0;
        aP=aW+aE+(Fe-Fw);
        Su=(p_ini(1,i)-p_ini(1,i+1))*Axv(1,i);
        d(1,i)=Axv(1,i)/aP;

        array(i,i-1)=-aW;
        array(i,i)=aP;
        array(i,i+1)=-aE;
        bound(i,1)=Su;

        residuals(1,i)=abs(aP*u_ini(1,i)-aW*u_ini(1,i-1)-Su);

    else
        Fw=rho*(u_ini(1,i-1)+u_ini(1,i))/2*Axp(1,i);
        Fe=mdot;
        aW=Fw;
        aE=0;
        aP=aW+aE+(Fe-Fw);
        Su=(p_ini(1,i)-p_ini(1,i+1))*Axv(1,i);
        d(1,i)=Axv(1,i)/aP;

        array(i,i-1)=-aW;
        array(i,i)=aP;
        bound(i,1)=Su;
        residuals(1,i)=abs(aP*u_ini(1,i)-aW*u_ini(1,i-1)-Su);
    end

end
    u=array\bound;%%%%new velocity%%%%%

    %%%%pressure corrector
    arrayp=zeros(nnode-2,nnode);
    boundp=zeros(nnode-2,1);
    for i=2:nnode-1
            aW=rho*d(1,i-1)*Axv(1,i-1);
            aE=rho*d(1,i)*Axv(1,i);
            Fw=rho*u(i-1)*Axv(1,i-1);
            Fe=rho*u(i)*Axv(1,i);
            aP=aW+aE;
            b=Fw-Fe;
            arrayp(i-1,i-1)=-aW;
            arrayp(i-1,i)=aP;
            arrayp(i-1,i+1)=-aE;
            boundp(i-1,1)=b;

    end
arrayp(:,[1,nnode])=0;%pressure correction at the first node and last node
is 0
pcorr=arrayp\boundp;%%correction pressure
p=p_ini+pcorr';%%corrected pressure
for i=1:nnodev
```

```
    u(i,1)=u(i,1)+d(1,i)*(pcorr(i,1)-pcorr(i+1,1));%corrected velocity
end
p(1,1)=p0-0.5*rho*(u(1,1)*Axv(1,1)/Axp(1,1))^2;%corrcted pressure at the
first node
%underrelaxation
u_ini=u'*und+u_ini*(1-und);
p_ini=p*und+p_ini*(1-und);
%%%%%%%
mdot=rho*u_ini(1,1)*Axv(1,1);
u_ini_p=mdot/rho./Axp;%u_ini_p updated
resi=sum(residuals(:));%sume of residuals
if resi<tor
    disp(j)
    break%simulation stops when resi is smaller than tor, and shows
iterations
end
end
disp(mdot)
xber= [0 : 0.01: 2];
Aber=0.5-(0.5-0.1)/L*xber;
mreal=((2*(p0-0)*(rho*AE)^2)/rho)^0.5;
preal=p0-(0.5*rho*mreal^2)./((rho*Aber).^2);
vreal=mreal./Aber/rho;
%plot(xdarrv,mdot,'x-',xdarrv,mreal,'.-')
plot(xdarrp,p_ini,'x-',xber,preal,'.-')
xlim([0 2])
figure;plot(xdarrv,u_ini,'x-',xber,vreal,'.-')
xlim([0 2])
```

Appendix B-6

```
%% 2D cone injection with random walk model
% Random walk model's isotropic behavior is applied in a cone injection.
% The injection describes the cone injection that is used in CFD model
% (ANSYS Fluent )
% Author: Kim Taewook
% Linkedin : https://www.linkedin.com/in/taewook-kim/
% GITHUB : github.com/Kimtaewookcode
% Email : kimtaewook87@gmail.com

clear
clf
clc
hold on;
steps = 100;
timestep=0.01;%s
trials=10;
x=zeros(trials, steps);
y=zeros(trials, steps);
J=0;
swirl=0;%0.3;%m/s
coneangle=13;%degree
conev=1.87;%m/s
xconev=conev*cosd(coneangle);
yconev=conev*sind(coneangle);
k=0.5;
k2=sqrt(k*2/3);
for t = 1:trials


for i=1:(steps-1)
J=rand;

if J<0.25
%x(t,i+1)=x(t,i)+(-k2*timestep);
%y(t,i+1)=y(t,i)+(k2*timestep);
x(t,i+1)=x(t,i)+(-k2+xconev)*timestep;
y(t,i+1)=y(t,i)+(k2+yconev)*timestep;


elseif J<0.5
%x(t,i+1)=x(t,i)+(k2*timestep);
%y(t,i+1)=y(t,i)+(k2*timestep);
x(t,i+1)=x(t,i)+(k2+xconev)*timestep;
y(t,i+1)=y(t,i)+(k2+yconev)*timestep;

elseif J<0.75
%x(t,i+1)=x(t,i)+(-k2*timestep);
%y(t,i+1)=y(t,i)+(-k2*timestep);
x(t,i+1)=x(t,i)+(-k2+xconev)*timestep;
y(t,i+1)=y(t,i)+(-k2+yconev)*timestep;

else
%x(t,i+1)=x(t,i)+(k2*timestep);
%y(t,i+1)=y(t,i)+(-k2*timestep);
x(t,i+1)=x(t,i)+(k2+xconev)*timestep;
y(t,i+1)=y(t,i)+(-k2+yconev)*timestep;


end
end
```

```
coneangle=coneangle-2.6;
xconev=conev*cosd(coneangle);
yconev=conev*sind(coneangle);
plot(x(t,:),y(t,:),'Color', [rand rand rand])
plot(x(t,100),y(t,100), 'ko')

%xlim([0 0.05])
%ylim([-0.05 0.05])
xlim([0 2])
ylim([-0.45 0.45])

end
%title('trajectories by instantaneous velocity(random walk) ');
title('trajectories by mean velocity+instantaneous velocity(random walk)
');
xlabel('X Displacement[m]');
ylabel('Y Displacement[m]');
```

Appendix B-7

```
/* MODIFIED SPHERICAL DRAG (REF:Clift R., Grace J.R., Weber M.E. Bubbles,
Drops, and Particl P112 )
ANSYS Fluent UDF
AUTHOR : KIMTAEWOOK
LINKEDIN : www.linkedin.com/in/kimtw
GITHUB : github.com/Kimtaewookcode
Email : kimtaewook87@gmail.com
*/
#include "udf.h"

DEFINE_DPM_DRAG(particle_drag_force, Re, p)
{
  real w, drag_force;

  if (Re < 0.01)
  {
    drag_force=9/64*Re+18.0;
    return (drag_force);
  }
  else if (Re < 20.0)
  {
    w = log10(Re);
    drag_force = 18.0 + 2.367*pow(Re,0.82-0.05*w) ;
    return (drag_force);
  }
  else if (Re < 260.0)
  {
    drag_force = 18.0 + 3.483*pow(Re,0.6305) ;
    return (drag_force);
  }
  else if (Re < 1500.0)
  {
    w = log10(Re);
    drag_force = 44.0048*pow(Re,-1.1242+0.1588*w) ;

  }
  else if (Re < 12000.0)
  {
    w = log10(Re);
    drag_force = 0.003491*3/4*pow(Re,3.5558-0.9295*w+0.1049*pow(w,2)) ;

  }
  else if (Re < 44000.0)
  {
    w = log10(Re);
    drag_force = 0.01207*3/4*pow(Re,1.6370-0.0636*w) ;

  }
  else if (Re < 338000.0)
  {
    w = log10(Re);
    drag_force = 0.00004581*3/4*pow(Re,2.5809-0.1546*w) ;

  }
  else if (Re < 400000.0)
  {
    w = log10(Re);
    drag_force = 3/4*pow(Re,29.78-5.3*w) ;

  }
```

```
    else if (Re < 1000000.0)
    {
      w = log10(Re);
      drag_force = 3/4*pow(Re,0.1*w-0.49) ;

    }
    else
    {
      w = log10(Re);
      drag_force = 3/4*(0.19*Re-80000) ;

    }
  }
```

Appendix B-8

```
/* TOMIYAMA'S DRAG
provided by ANSYS CUSTOMER PORTAL
*/


#include "udf.h"


#define drag_surface_tension 0.072 /* N/m */

real
calc_cap_drag(real Re)
{
return 72./ Re;
}


real
calc_sphere_drag(real Re)
{
return 24.*(1.+0.15*pow(Re,0.687))/Re;
}


real
calc_ellipse_drag(Tracked_Particle *p)
{
cphase_state_t *c = &(p->cphase); /* cell information at particle
location*/
real drag;
real Eo;

Eo = 9.81*(c->rho - P_RHO(p)) * SQR(P_DIAM(p)) / drag_surface_tension;

drag = 8./3. * Eo / (Eo + 4.);

return drag;
}

DEFINE_DPM_DRAG(particle_drag_tomiyama,Re,p)
{

real drag_coef;
real CD_sphere,CD_cap,CD_ellipse;


CD_cap = calc_cap_drag(Re);

CD_sphere = calc_sphere_drag(Re);

CD_ellipse = calc_ellipse_drag(p);

drag_coef =  MAX(MIN(CD_sphere,CD_cap),CD_ellipse);

return (18.*drag_coef*Re/24.);

}
```

Appendix C<Boundary condition-OpenFoam>

Appendix C-1< Alpha.water>

```
/*--------------------------------*- C++ -*----------------------------------
--*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFoam: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1606+                                |
|   \\  /    A nd           | Web:      www.OpenFoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------
--*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      alpha.water;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* //

dimensions      [0 0 0 0 0 0 0];

internalField   nonuniform List<scalar>
939360
(0
0
0
.
.
.
.
1
1
1
1
)
;

boundaryField
{
    outlet
    {
        type            inletOutlet;
        inletValue       uniform 0;
        value           uniform 0;
    }
    wall
    {
        type            zeroGradient;
    }
    inlet
    {
        type            uniformFixedValue;
        uniformValue     constant 1;
        value           uniform 1;
    }
}
```

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

Appendix C-2< epsilon>

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFoam: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1606+                                |
|   \\  /    A nd           | Web:      www.OpenFoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      epsilon;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -3 0 0 0 0];

internalField   uniform 0.001;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           $internalField;
    }

    outlet
    {
        type            inletOutlet;
        inletValue      $internalField;
        value           $internalField;
    }
    wall
    {
        type            epsilonWallFunction;
        value           $internalField;
    }

}

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

Appendix C-3< k>

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFoam: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1606+                                |
|   \\  /    A nd           | Web:      www.OpenFoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      k;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform 1e-3;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           $internalField;
    }

    outlet
    {
         type            inletOutlet;
        inletValue       $internalField;
        value           $internalField;
    }
    wall
    {
        type            kqRWallFunction;
        value           $internalField;
    }

}


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

Appendix C-4< nut>

```
/*--------------------------------*- C++ -*----------------------------------
--*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFoam: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1606+                                |
|   \\  /    A nd           | Web:      www.OpenFoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------
--*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      nut;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* //

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

internalField   uniform 0;

boundaryField
{

    inlet
    {
        type            calculated;
        value           uniform 0;
    }

    outlet
    {
        type            calculated;
        value           uniform 0;
    }
    wall
    {
        type            nutkWallFunction;
        value           uniform 0;
    }
}


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

Appendix C-5< p_rgh>

```
/*---------------------------------*- C++ -*----------------------------------
--*\
| =========                 |                                                |
| \\      /  F ield         | OpenFoam: The Open Source CFD Toolbox          |
|  \\    /   O peration     | Version:  v1606+                               |
|   \\  /    A nd           | Web:      www.OpenFoam.com                     |
|    \\/     M anipulation  |                                                |
\*----------------------------------------------------------------------------
--*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p_rgh;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* //

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

internalField      uniform 0;

boundaryField
{
    inlet
    {
        type            fixedFluxPressure;
        value           uniform 0;
    }

    outlet
    {
        type            prghTotalPressure;
        p0              uniform 0;
        value           uniform 0;
    }
    wall
    {
        type            fixedFluxPressure;
        value           uniform 0;
    }

}

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

Appendix C-6< Uair>

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFoam: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1606+                                |
|   \\  /    A nd           | Web:      www.OpenFoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      binary;
    class       volVectorField;
    object      Uair;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform (0 0 0);

boundaryField
{
    inlet
    {
        type            uniformFixedValue;
        uniformValue      table
                        (
                            (0  (0 0 1.868))
                            (20  (0 0 1.868))
                        );
    }

    outlet
    {
        type            pressureInletOutletVelocity;
        value           $internalField;
        inletValue       $internalField;
    }
    wall
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }

}
//
**************************************************************************
//
```

Appendix D<Constant-OpenFoam>

Appendix D-1<g>

```
/*--------------------------------*- C++ -*----------------------------------
--*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFoam: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1606+                                |
|   \\  /    A nd           | Web:      www.OpenFoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------
--*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedVectorField;
    location    "constant";
    object      g;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* //

dimensions      [0 1 -2 0 0 0 0];
value           ( 0 0 -9.81);


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

Appendix D-2< particleProperties >

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFoam: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1606+                                |
|   \\  /    A nd           | Web:      www.OpenFoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      particleProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solution
{
    active          true;
    coupled         true;
    transient       yes;
    cellValueSourceCorrection no;

    maxCo           0.3;

    interpolationSchemes
    {
        rho         cell;
        U           cellPoint;
        mu          cell;
        gradAlpha   cellPoint;
    }

    averagingMethod dual;

    integrationSchemes
    {
        U               Euler;
    }

    sourceTerms
    {
        schemes
        {
            U           semiImplicit 0.8;
        }
    }
}

constantProperties
{
    rho0            1.225;//changed
    alphaMax        0.9;//needtobechanged
}

subModels
```

```
{
   particleForces
   {
      sphereDrag;
      //WenYuDrag
      //{
      //   alphac     alphac;
      //}
      gravity;
      interface
      {
         C          -10;
         alphaName   alpha.water;
      }
   }

   injectionModels
   {
      model1
      {
         type           patchInjection;
         massTotal       0.416e1;
         SOI            0;
         parcelBasisType fixed;//mass;
         nParticle       1;
         patchName       inlet;
         duration        20;
         parcelsPerSecond 1e3;//1e5;
         U0             (0 0 1.868);//changed
         flowRateProfile constant 1;
         sizeDistribution
         {
            type       RosinRammler;

            RosinRammlerDistribution
            {
               minValue       0.0018;
               maxValue       0.0847;
               d             0.0054;//droplet diameter that has the
largest probability
               n             3.5;//from fluent
            }
         }
   }
   }
   dispersionModel stochasticDispersionRAS;//none;addedlibrary and
recompiled

   patchInteractionModel localInteraction;

   localInteractionCoeffs
   {
      patches
      (
         wall
         {
            type rebound;
            e    0.95;
            mu   0.09;
         }
         //base
```

```
        //{
        //    type rebound;
        //    e    0.95;
        //    mu   0.09;
        //}
        inlet
        {
           type escape;
        }
        outlet
        {
           type escape;
        }
    );
}

heatTransferModel none;

surfaceFilmModel none;

packingModel implicit;

explicitCoeffs
{
   particleStressModel
   {
      type HarrisCrighton;
      alphaPacked 0.6;
      pSolid 10.0;
      beta 2.0;
      eps 1.0e-7;
   }
   correctionLimitingMethod
   {
      type absolute;
      e 0.9;
   }
}

implicitCoeffs
{
   alphaMin 0.001;
   rhoMin 1.0;
   applyGravity false;
   applyLimiting   false;
   particleStressModel
   {
      type HarrisCrighton;
      alphaPacked 0.9;
      pSolid 5.0;
      beta 2.0;
      eps 1.0e-2;
   }
}

dampingModel relaxation;

relaxationCoeffs
{
   timeScaleModel
   {
```

```
            type nonEquilibrium;
            alphaPacked 0.7;
            e 0.8;
        }
    }

    isotropyModel stochastic;

    stochasticCoeffs
    {
        timeScaleModel
        {
            type isotropic;
            alphaPacked 0.7;
            e 0.8;
        }
    }

    stochasticCollisionModel none;

    radiation off;
}


cloudFunctions
[1]


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

Appendix D-3< transportProperties >

```
/*--------------------------------*- C++ -*----------------------------------
--*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFoam: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1606+                                |
|   \\  /    A nd           | Web:      www.OpenFoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------
--*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* //

phases (water air);

water
{
    transportModel  Newtonian;
    nu              nu [ 0 2 -1 0 0 0 0 ] 1e-06;
    rho             rho [ 1 -3 0 0 0 0 0 ] 1000;
    CrossPowerLawCoeffs
    {
        nu0             nu0 [ 0 2 -1 0 0 0 0 ] 1e-06;
        nuInf           nuInf [ 0 2 -1 0 0 0 0 ] 1e-06;
        m               m [ 0 0 1 0 0 0 0 ] 1;
        n               n [ 0 0 0 0 0 0 0 ] 0;
    }

    BirdCarreauCoeffs
    {
        nu0             nu0 [ 0 2 -1 0 0 0 0 ] 0.0142515;
        nuInf           nuInf [ 0 2 -1 0 0 0 0 ] 1e-06;
        k               k [ 0 0 1 0 0 0 0 ] 99.6;
        n               n [ 0 0 0 0 0 0 0 ] 0.1003;
    }
}

air
{
    transportModel  Newtonian;
    nu              nu [ 0 2 -1 0 0 0 0 ] 1.48e-05;
    rho             rho [ 1 -3 0 0 0 0 0 ] 1.225;//1;
    CrossPowerLawCoeffs
    {
        nu0             nu0 [ 0 2 -1 0 0 0 0 ] 1e-06;
        nuInf           nuInf [ 0 2 -1 0 0 0 0 ] 1e-06;
        m               m [ 0 0 1 0 0 0 0 ] 1;
        n               n [ 0 0 0 0 0 0 0 ] 0;
    }

    BirdCarreauCoeffs
    {
        nu0             nu0 [ 0 2 -1 0 0 0 0 ] 0.0142515;
        nuInf        nuInf [ 0 2 -1 0 0 0 0 ] 1e-06;
        k           k [ 0 0 1 0 0 0 0 ] 99.6;
        n           n [ 0 0 0 0 0 0 0 ] 0.1003;
    }
}

sigma          sigma [ 1 0 -2 0 0 0 0 ] 0.07;


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

Appendix D-4< `turbulenceProperties` >

```
/*--------------------------------*- C++ -*----------------------------------
--*\
| =========                 |                                                |
| \\      /  F ield         | OpenFoam: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1606+                                |
|   \\  /    A nd           | Web:      www.OpenFoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------
--*/
FoamFile
{
   version     2.0;
   format      ascii;
   class       dictionary;
   location    "constant";
   object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* //

simulationType  RAS;

RAS
{
   RASModel        kEpsilon;
   turbulence      on;
   printCoeffs     on;
}
//
*************************************************************************
//
```

Appendix E<system-OpenFoam>

Appendix E-1< controlDict >

```
/*--------------------------------*- C++ -*----------------------------------
--*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFoam: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1606+                                |
|   \\  /    A nd           | Web:      www.OpenFoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------
--*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* //

application     MPPICInterFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         20.0;

deltaT          0.01;

writeControl    timeStep;//adjustableRunTime;

writeInterval   50;//1;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;//uncompressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

adjustTimeStep  off;

maxCo           1.0;
maxAlphaCo      1.0;

maxDeltaT       0.05;
```

```
functions
{

   // minMax
    //{
    //  type fieldMinMax;
    //  functionObjectLibs ("libfieldFunctionObjects.so");
    //  outputControl timeStep; //outputTime;
    //  fields (U);
   // }
}

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

Appendix E-2< createPatchDict >

```
/*--------------------------------*- C++ -*----------------------------------
--*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFoam: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1606+                                |
|   \\  /    A nd           | Web:      www.OpenFoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------
--*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      createPatchDict;
}

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

pointSync false;

// Patches to create.
patches
(
    {
        // Name of new patch
        name inlet;

        // Type of new patch
        patchInfo
        {
            type patch;
        }

        // How to construct: either from 'patches' or 'set'
        constructFrom set;

        // If constructFrom = set : name of faceSet
        set inlet;
    }
);

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

Appendix E-3< `decomposeParDict` >

```
/*--------------------------------*- C++ -*----------------------------------
--*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFoam: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1606+                                |
|   \\  /    A nd           | Web:      www.OpenFoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------
--*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      decomposeParDict;
}

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

numberOfSubdomains 4;

method          hierarchical;

hierarchicalCoeffs
{
    n               (1 1 4);
    delta           0.001;
    order           xyz;
}


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

Appendix E-4< `fvSchemes` >

```
/*--------------------------------*- C++ -*----------------------------------
--*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFoam: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1606+                                |
|   \\  /    A nd           | Web:      www.OpenFoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------
--*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default         none;
    div(rhoPhi,U)             Gauss limitedLinearV 1;//Gauss upwind;
    div(phi,alpha)            Gauss vanLeer;
    div(phirb,alpha)          Gauss linear;
    div(alphaRhoPhic,k)       Gauss upwind;
    div(alphaRhoPhic,epsilon)   Gauss upwind;
    div(((alphac*rho)*nuEff)*dev2(T(grad(U)))) Gauss linear;
    div(phiGByA,kinematicCloud:alpha) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

fluxRequired
```

```
{
   default        no;
   p_rgh;
   pcorr;
   alpha.water;
}


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

Appendix E-5< fvSolution >

```
/*--------------------------------*- C++ -*----------------------------------
--*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFoam: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1606+                                |
|   \\  /    A nd           | Web:      www.OpenFoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------
--*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* //

solvers
{
    "alpha.water.*"
    {
        nAlphaCorr      2;
        nAlphaSubCycles 2;
        cAlpha          1;

        MULESCorr       yes;
        nLimiterIter    2;

        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-7;//1e-7;
        relTol          0;
        maxIter         1000;
    }

    pcorr
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-5;
        relTol          0;
    }

    p_rgh
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-07;
        relTol          0.05;
    }

    p_rghFinal
    {
        $p_rgh;
        relTol          0;
    }

    "(U|k|epsilon).*"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-06;
        relTol          0;
    }

    kinematicCloud:alpha
    {
        solver          GAMG;
        tolerance       1e-05;
        relTol          0.1;
        smoother        GaussSeidel;
        cacheAgglomeration true;
        nCellsInCoarsestLevel 10;
        agglomerator    faceAreaPair;
        mergeLevels     1;
    }
}

PIMPLE
{
    momentumPredictor   no;
    nOuterCorrectors    1;
    nCorrectors         3;
    nNonOrthogonalCorrectors 0;
}

relaxationFactors
{
    equations
    {
        ".*"               1;
    }
}

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

Appendix E-6< `setFieldsDict` >

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFoam: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1606+                                |
|   \\  /    A nd           | Web:      www.OpenFoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
    volScalarFieldValue alpha.water 0
);

regions
(
    boxToCell
    {
        box (-4.5 -3.5 0) (4.5 3.5 6.67);
        fieldValues
        (
            volScalarFieldValue alpha.water 1
        );
    }
);


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

Appendix E-7< `topoSetDict` >

```
/*--------------------------------*- C++ -*----------------------------------
--*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFoam: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v1606+                                |
|   \\  /    A nd           | Web:      www.OpenFoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------
--*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      topoSetDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* //

actions
(
    {
        name    inlet;
        type    faceSet;
        action  new;
        source  boxToFace;
        sourceInfo
        {
            box (-0.17 -0.17 -0.03)(0.17 0.17 0.001);
        }
    }

    {
        name    inletZone;
        type    faceZoneSet;
        action  new;
        source  setToFaceZone;
        sourceInfo
        {
            faceSet    inlet;
        }
    }
);

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