POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

BUMP - BUbble column Multiphase Project

Stefano Passoni, Martina Di Gennaro, Riccardo Giordani

Contents

---

1. Introduction

Bubble columns are multiphase reactors where the dispersed phase (gas) is introduced into a stationary or flowing liquid (continuous phase) and provide a good experimental setup to study the turbulent phenomena in dense bubbly flows. The functioning is apparently simple, as the ascending gas-phase creates a buoyancy-driven flow inducing the recirculation of the liquid phase. The local and the global fluid dynamic properties are related to the prevailing flow regime which may be homogeneous (bubbly flow condition at low superficial gas velocity, tipically $U_G < 0.03\ m/s$) or heterogeneous (churn-turbulent flow condition, $U_G > 0.1\ m/s$, $\alpha_G \geq 0.3$). The homogeneous flow regime can be classified into the *mono-dispersed homogeneous* flow regime, characterized by a mono-dispersed bubble size distribution (BSD), and the *pseudo-homogeneous* flow regime, characterized by a poly-dispersed BSD. The distinction between mono-dispersed and poly-dispersed BSDs is based upon the change in the sign of the lift force coefficient.
Numerous industrial designs have been based on empirical correlations, but such approaches remain somewhat limited when increasing the reactor performance is sought. To improve the description of the physical phenomenon occurring in bubble columns a promising numerical method is the Computational Fluid Dynamics (CFD). CFD helps to understand the complex two-phase fluid dynamics in the bubble column through details of mean flows (fields of three components of mean velocities and mean gas hold-up), interphase rates of mass, energy and momentum transfer and turbulence parameters (such as turbulent kinetic energy, energy dissipation rate, Reynolds stresses, etc.).
In the present work, bubble column is numerically simulated by using the open source CFD tool OpenFOAM []. More in detail, the objective of the present work is the validation of the already existing *multiphaseEulerFoam* solver working for the case of *mono-dispersed homogeneous* bubble column. For the case study we have considered a multiphase mixture of air and water without taking into account the heat transfer phenomena.
The *multiphaseEulerFoam* is used for simulation of incompressible multiphase flows in vertical pipelines. This model combines the Eulerian-Eulerian (two-fluid).
The structure of the report is as follows. In Section 2, the experimental benchmark taken as reference is illustrated. In Section 3,

2. Experimental Benchmark

The experimental benchmark taken as reference for the validation comes from the literature, in particular from the study of Krepper et al. []. The experimental setup consists of three rectangular channels 20cm^2 in cross-section ($0.1 \times 0.02\ m$) bolted together at the flanges resulting in a bubble column of 1-meter height (fig 1.). The test facility is initially filled with water up to a specified (..) height. In the bottom of the test section is present a rectangular porous stone adopted as gas sparger $0.02m$ wide, $0.01m$ depth and $0.01m$ height.This sparger produced a *mono-dispersed homogeneous* bubble size distribution with a bubble diameter of 3-5 mm. The superficial gas flow rate, U_G, is varied between three different value, 0.006 m/s, 0.008 m/s and 0.010 m/s, in the *mono-dispersed homogenous* flow regime. The superficial velocity is calculated from the measured volume flow rates using:

$$U_G = \frac{Q_G}{A_{CS}} \tag{1}$$

where Q_G is the measured volume flow rate of the gas and A_{CS} is the cross-sectional area of the test section. The column operates in the dispersed bubble bubbly regime, characterized by the absence of bubble coalescence or breakup, for the superficial gas velocities adopted in this work. For each superficial gas velocity the volume average void fraction, α_V, also known as *gas holdup*, can be calculated using:

$$\alpha_V = \frac{h_{final} - h_{initial}}{h_{final}} \tag{2}$$

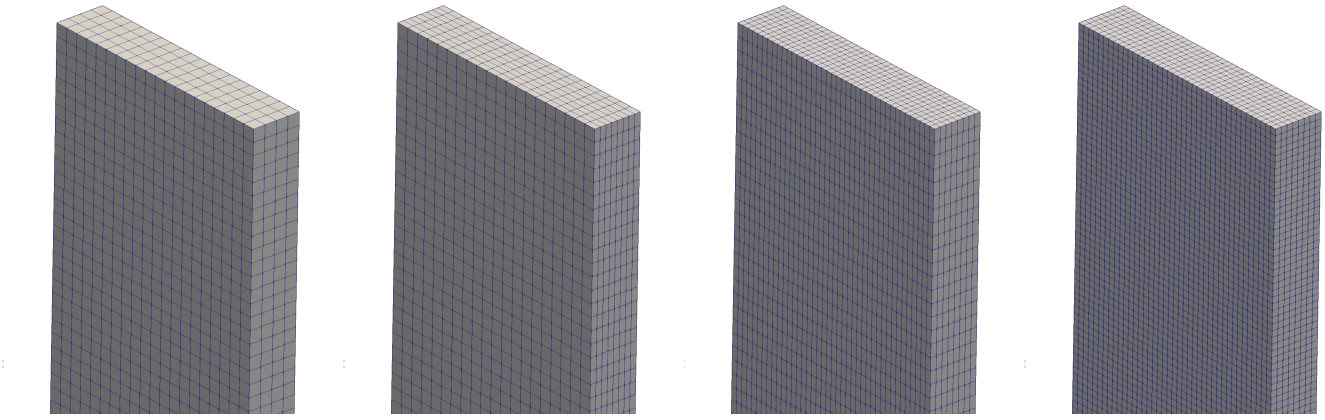
where h_{final} is the heigth of the liquid column after the column has been aerated and $h_{initial}$ is the stationary height of the liquid column before aeration. The experimental data consist of wire-mesh local void fraction measurements performed at two different axial heights y = 0.08 m and y = 0.63 m above the gas sparger). The wire mesh sensor is placed in the cross-section by bolting it to two flanged rectangular channels. The wire-mesh sensor data was recorded at a frequency of 2500 Hz for a time period of 60 s.

3. Numerical Setup

In this section all the details related to the numerical setup will be presented and discussed.

3.1. Domain and Mesh

The numerical domain has been modeled and meshed with the utility `blockMesh` according to the size of the experimental facility reported in section 2. The resulting mesh is structured and made of only regular hexahedral elements. Four different refinement levels were analyzed for the purpure of a grid sensitivity analysis. Figure 1 displays a section of the generated meshes and their details are instead summarized in table 1. According to [1], the mesh size should be 1.5 times the size of the bubbles. Therefore, given the size distribution found in the experiments (see section 2, the coarsest mesh should be the most adequate since it has a cell size of 5mm in each direction. The grid refinement study has the purpose of verify if such a corse mesh has negative influence on the fluid-dynamic phenomena inside the bubble column.
Speaking about the inlet section, the porous stone used as gas sparger in the experiments has been modeled as a zero-thickness surface having the size of the sparger itself, namely a 0.02x0.01 m^2 cross-section.



(a) Mesh A (b) Mesh B (c) Mesh C (d) Mesh D

Figure 1: Different mesh densities considered in the analysis

	N_x	N_y	N_z	Total no. of elements
Mesh A	20	200	4	16000
Mesh B	20	200	8	32000
Mesh C	40	200	8	64000
Mesh D	40	400	8	128000

Table 1: Mesh size details

3.2. Boundary Conditions

The list of boundary conditions applied to the domain is summarized in table 2. At the inlet, three different gas superficial velocities were applied according to the three cases studied, respectively 6, 8 and 10 mm/s. These superficial velocities are an experimental input and were calculated as the volumetric flow rate of gas injected in the column divided by its volume. In OpenFOAM, a`flowRateInletVelocity` boundary condition was employed. This requires a mass flow rate input and a reference density in order to calculate the inlet velocity. The chosen density was 1.185 kg/m^3 which corresponds to air density at ambient conditions of 25°C and 1 atm.

Moreover, a custom `fvModels` boundary condition was developed to implement a degassing boundary condition. The complete listing of the source code is provided in Appendix A. Typically, a this type boundary condition is used to model a free surface through which dispersed gas bubbles are allowed to escape, but the continuous phase is not. A typical application is a bubble column in which the user want to reduce computational cost by not including the freeboard region in the simulation [2]. This boundary conditions offers also an improvement in terms of numerical stability since the water-air interface is not modeled. When the degassing boundary

condition is specified for an outlet, the continuous liquid phase sees the boundary as a free-slip wall and does not leave the domain. The dispersed gas phase sees the boundary as an outlet. The code provided calculates an implicit mass source term to remove gas phase from the cells adjacent to the outlet boundary and also computes a degassing force to account for the effect of gas removal on the continuous phase.

Variable	Inlet	Outlet	Walls
alpha.air	fixedValue	zeroGradient	zeroGradient
alpha.water	fixedValue	zeroGradient	zeroGradient
k.water	fixedValue	inletOutlet	kqRWallFunction
epsilon.water	fixedValue	inletOutlet	epsilonWallFunction
omega.water	fixedValue	inletOutlet	omegaWallFunction
km	fixedValue	inletOutlet	kqRWallFunction
epsilonm	fixedValue	inletOutlet	epsilonWallFunction
nut.water	calculated	calculated	nutkWallFunction
p	calculated	calculated	calculated
U.air	flowRateInletVelocity	pressureInletOutletVelocity	fixedValue
U.water	fixedValue	slip	fixedValue

Table 2: List of boundary conditions applied to the three different boundaries

3.3. Turbulence modeling

Mettiamo qui la parte di equazioni sui modelli di turbolenza che sta scrivendo Riccardo? Cosi dopo mettiamo solo i risultati.

3.4. Solver

3.5. Multiphase Forces

4. Results and Discussion

4.1. CFD methodology

4.2. Mesh sensitivity

		N_x	N_y	N_z
run001	**Mesh A**	20	200	4
run002	**Mesh B**	20	200	8
run003	**Mesh C**	40	200	8
run004	**Mesh D**	40	400	8

Table 4: Meshes

	$J_{in}[mm/s]$	Mesh	Time step	Turbulence model	Drag model	Lift model	Turbulent dispersion model	Wall lubrication model
run001	10	A	0.005	kOmegaSSTSato	IshiiZuber	Tomiyama	None	None
run002	10	B	0.005	kOmegaSSTSato	IshiiZuber	Tomiyama	None	None
run003	10	C	0.005	kOmegaSSTSato	IshiiZuber	Tomiyama	None	None
run004	10	D	0.005	kOmegaSSTSato	IshiiZuber	Tomiyama	None	None
run005	10	A	0.010	kOmegaSSTSato	IshiiZuber	Tomiyama	None	None
run006	10	A	0.0025	kOmegaSSTSato	IshiiZuber	Tomiyama	None	None
run007	10	A	0.005	LaheyKEpsilon	IshiiZuber	Tomiyama	None	None
run008	10	A	0.005	mixtureKEpsilon	IshiiZuber	Tomiyama	None	None
run009	10	A	0.005	kOmegaSSTSato	IshiiZuber	None	None	None
run010	10	A	0.005	kOmegaSSTSato	IshiiZuber	Moraga	None	None
run011	10	A	0.005	kOmegaSSTSato	IshiiZuber	Tomiyama	Burns	None
run012	10	A	0.005	kOmegaSSTSato	IshiiZuber	Tomiyama	LopezDeBertodanonone	None
run013	10	A	0.005	kOmegaSSTSato	IshiiZuber	Tomiyama	Burns	Antal
run014	10	A	0.005	kOmegaSSTSato	IshiiZuber	Tomiyama	Burns	Frank
run015	8	A	0.005	kOmegaSSTSato	IshiiZuber	Tomiyama	Burns	Antal
run016	6	A	0.005	kOmegaSSTSato	IshiiZuber	Tomiyama	Burns	Antal

Table 3: Runs performed

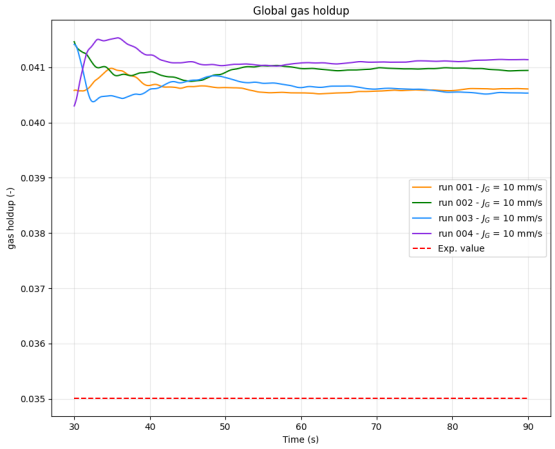
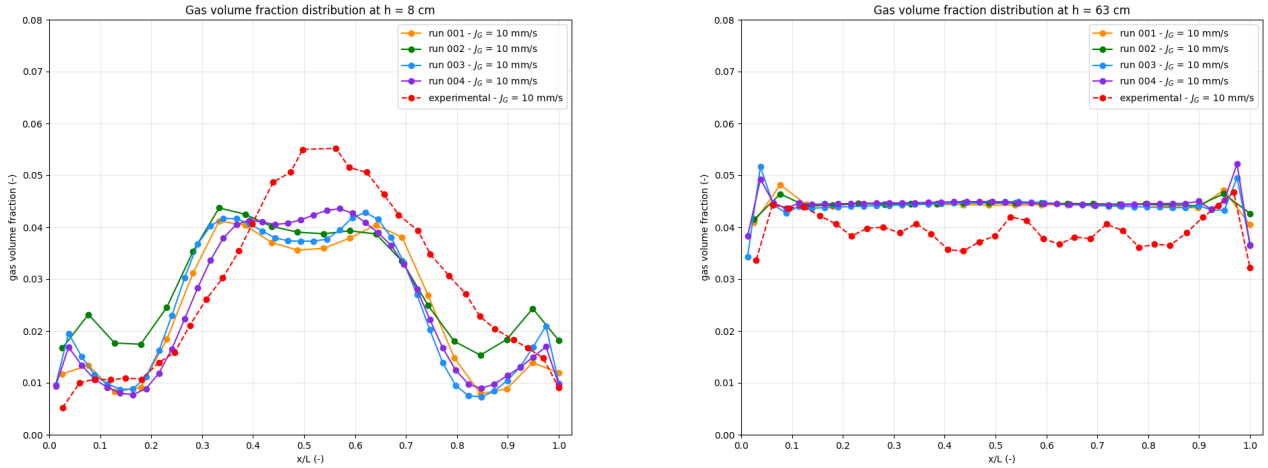


Figure 2: Averaged holdup for different meshes



(a) h = 8 cm

(b) h = 63 cm

Figure 3: Gas volume fraction horizontal distribution at different heights for different meshes

4.3. Time sensitivity

	Δt
run006	0.0025
run001	0.005
run005	0.010

Table 5: Time steps

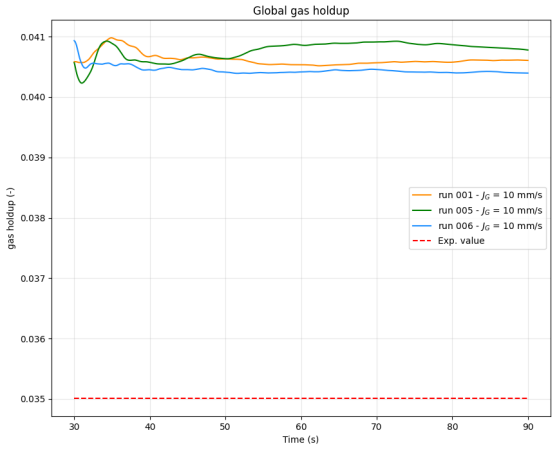
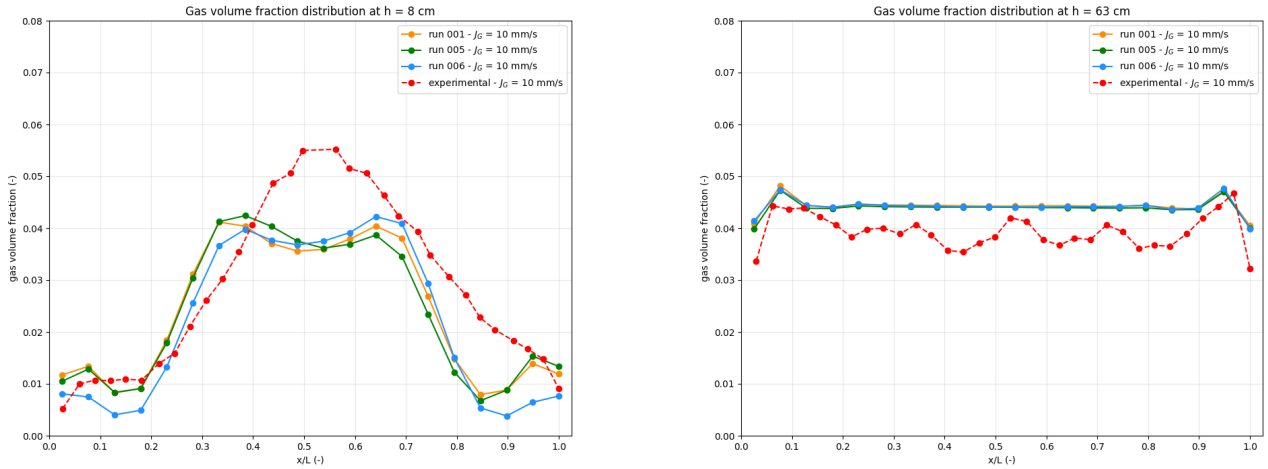


Figure 4: Averaged holdup for different time steps



(a) h = 8 cm

(b) h = 63 cm

Figure 5: Gas volume fraction horizontal distribution at different heights for different time steps

4.4. Turbulence models

4.4.1 $k-\omega$ SST Sato

The $k-\omega$ *SST Sato* model [3] has originally developed to tackle the problem of heat transfer for boundary layer flows. It combines eddy-viscosity models for the momentum equations and eddy-diffusivity model for heat transer.
The idea behind SST model is to combine the best elements of the $k-\epsilon$ model and the $k-\omega$ model with the help of a blending function. Indeed, the $k-\epsilon$ model presents deficiencies near the wall, where fine meshes and specific near-wall treatments are required. Whereas, the $k-\omega$ model is strongly dependent of the solution to free stream values of ω outside the boundary layer.

$$\frac{\partial \rho k}{\partial t} + \frac{\partial \rho U_j k}{\partial x_j} = \tilde{P}_k - \beta^\star \rho \omega k + \frac{\partial}{\partial x_j}\left(\Gamma_k \frac{\partial k}{\partial x_j}\right) \tag{3}$$

$$\frac{\partial \rho \omega}{\partial t} + \frac{\partial \rho U_j \omega}{\partial x_j} = \frac{\gamma}{\nu_t} P_k - \beta \rho \omega^2 + \frac{\partial}{\partial x_j}\left(\Gamma_\omega \frac{\partial \omega}{\partial x_j}\right) + (1-F_1) 2\rho\sigma_{\omega 2} \frac{1}{\omega} \frac{\partial k}{\partial x_j} \frac{\partial \omega}{\partial x_j} \tag{4}$$

with

$$\Gamma_k = \mu + \frac{\mu_t}{\sigma_k},\ \Gamma_\omega = \mu + \frac{\mu_t}{\sigma_\omega},\ P_k = \tau_{ij} \frac{\partial U_i}{\partial x_j},\ \tilde{P}_k = \min(P_k; c_l \epsilon),\ \mu_t = \rho \frac{a_1 k}{\max(a_1 \omega; S \cdot F_2)} \tag{5}$$

The coefficients, ϕ of the model are function of F_1: $\phi = F_1\phi_1 + (1 - F_1)\phi_2$, where ϕ_1 and ϕ_2 stand for the coefficients fo the $k - \omega$ and the $k - \epsilon$ model rispectively:

$$\sigma_{k1} = 1.176,\ \sigma_{\omega 1} = 2.000,\ \kappa = 0.41,\ \gamma_1 = 0.5532,\ \beta_1 = 0.0750,\ \beta^\star = 0.09,\ c_1 = 10 \tag{6}$$

$$\sigma_{k2} = 1.00,\ \sigma_{\omega 2} = 1.168,\ \kappa = 0.41,\ \gamma_2 = 0.4403,\ \beta_2 = 0.0828,\ \beta^\star = 0.09 \tag{7}$$

with

$$F_1 = \tanh \tag{8}$$

Here, the absolute value of the strain rate, S, is used in the definition of the eddy viscosity instead of the vorticity in order to increase the generality of the method beyond aerodynamic applications.
The turbulent heat flux vector is modelled with the help of a turbulent diffusivity:

4.4.2 Lahey $k - Epsilon$

4.4.3 Mixture $k - Epsilon$

	Turbulence model
run001	kOmegaSSTSato
run007	LaheyKEpsilon
run008	mixtureKEpsilon

Table 6: Turbulence models

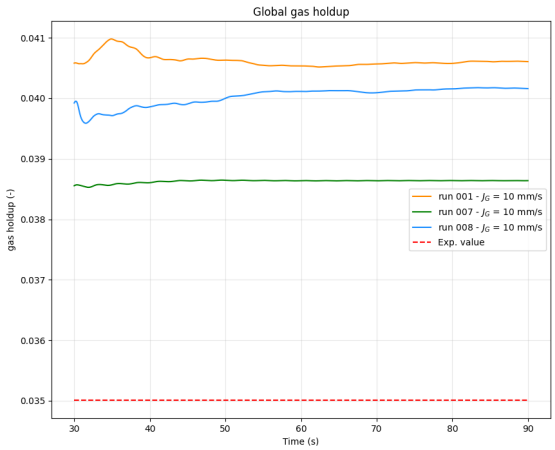
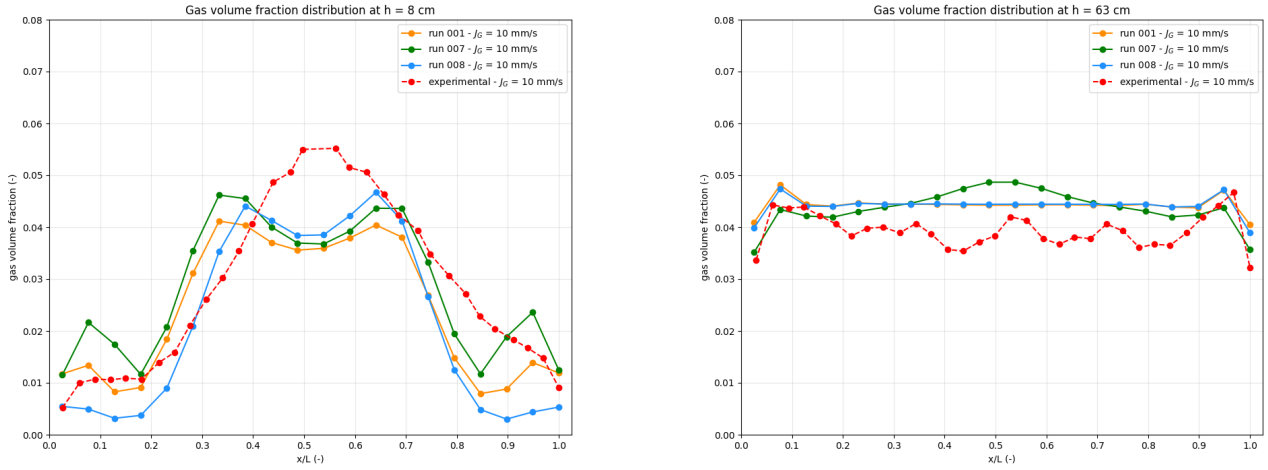


Figure 6: Averaged holdup for different turbulence models



(a) h = 8 cm (b) h = 63 cm

Figure 7: Gas volume fraction horizontal distribution at different heights for different turbulence models

4.5. Drag models

4.5.1 IshiiZuber

	Drag model
all runs	

Table 7: Drag models

4.6. Lift models

4.6.1 Tomiyama

4.6.2 Moraga

	Lift model
run009	None
run001	Tomiyama
run010	Moraga

Table 8: Lift models

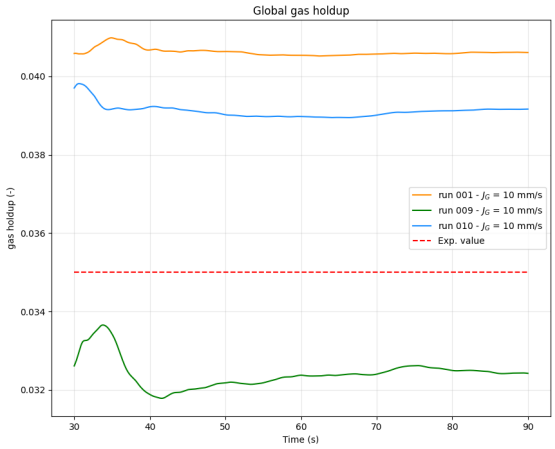
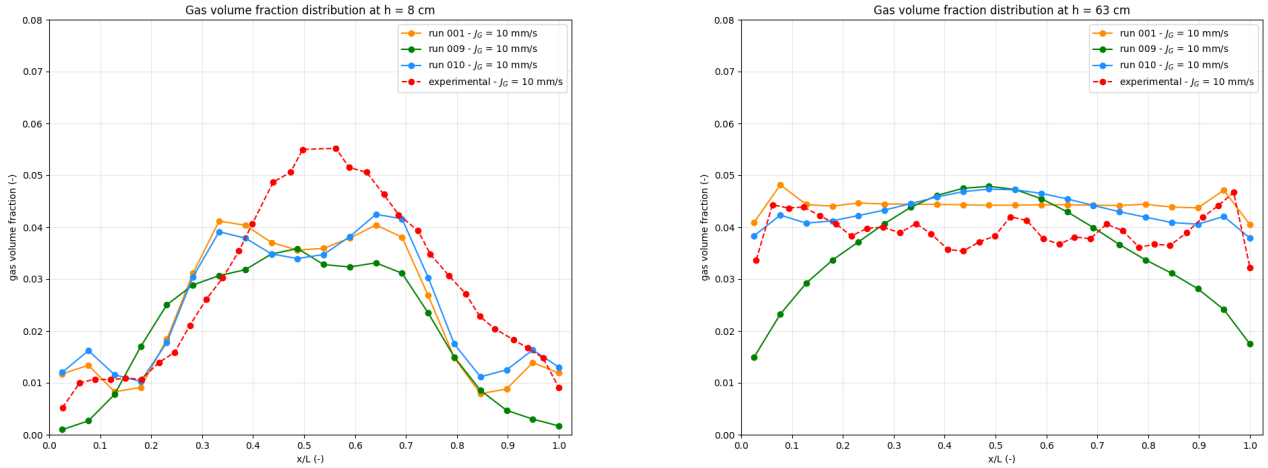


Figure 8: Averaged holdup for different lift models



(a) h = 8 cm

(b) h = 63 cm

Figure 9: Gas volume fraction horizontal distribution at different heights for different lift models

4.7. Turbulent dispersion models

4.7.1 Burns

4.7.2 LopezDeBertodanonone

	Turbulent dispersion model
run001	None
run011	Burns
run012	LopezDeBertodanonone

Table 9: Turbulent dispersion models

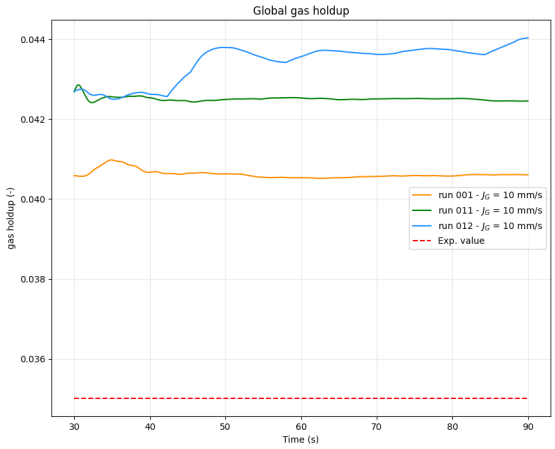
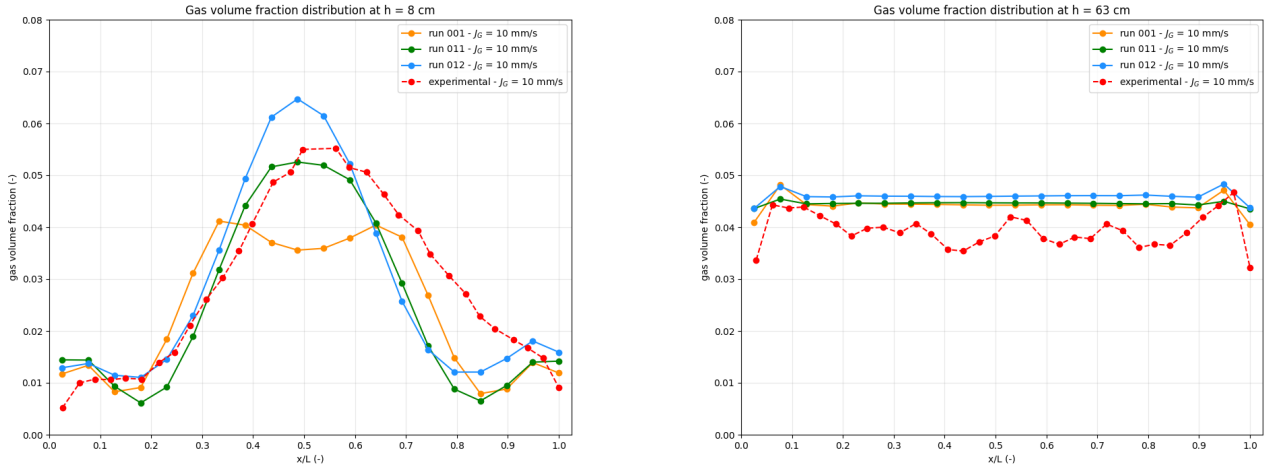


Figure 10: Averaged holdup for different turbulent dispersion models



(a) h = 8 cm

(b) h = 63 cm

Figure 11: Gas volume fraction horizontal distribution at different heights for different turbulent dispersion models

4.8. Wall lubrication models

4.8.1 Antal

4.8.2 Frank

	Wall lubrication model
run001	None
run013	Antal
run014	Frank

Table 10: Wall lubrication models

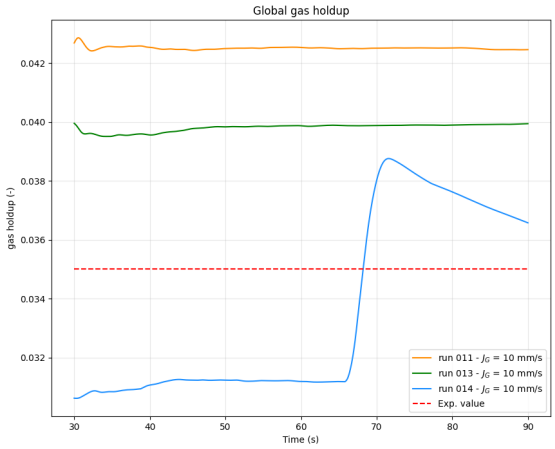
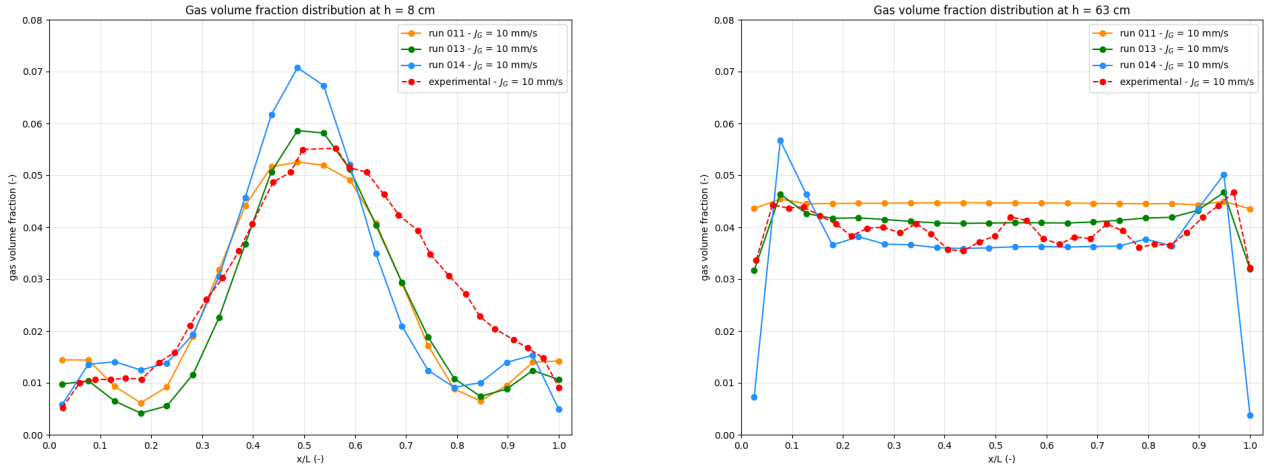


Figure 12: Averaged holdup for different wall lubrication models



(a) h = 8 cm

(b) h = 63 cm

Figure 13: Gas volume fraction horizontal distribution at different heights for different wall lubrication models

4.9. Inflow velocity

	$J_{in}[mm/s]$
run013	10
run015	8
run016	6

Table 11: Inflow velocities

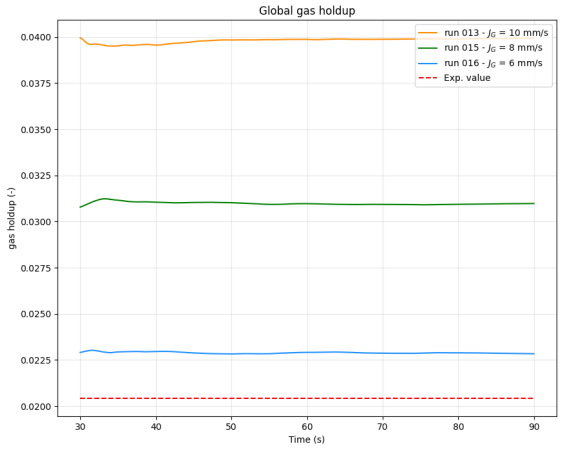
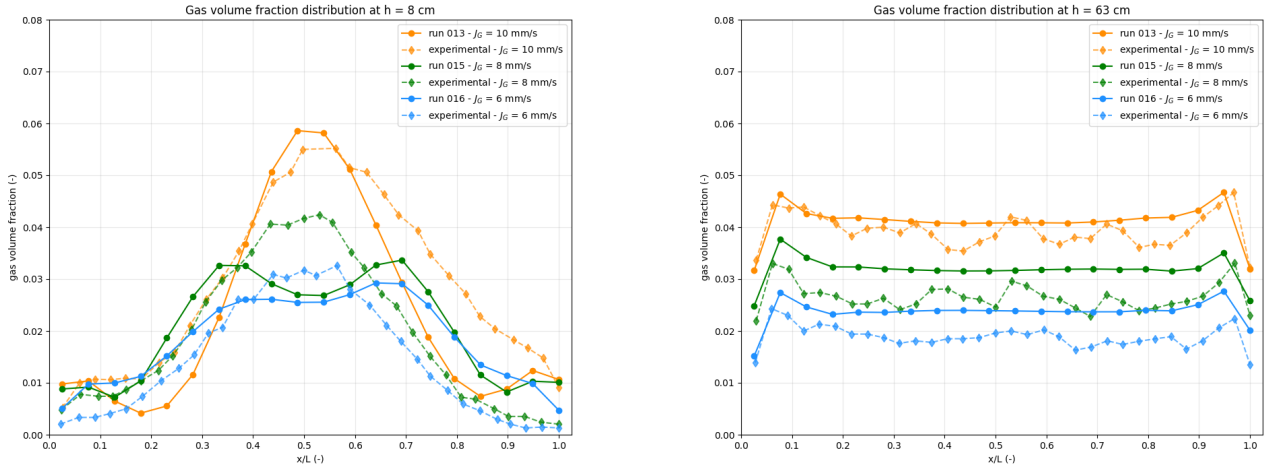


Figure 14: Averaged holdup for different inflow velocities



(a) h = 8 cm

(b) h = 63 cm

Figure 15: Gas volume fraction horizontal distribution at different heights for different inflow velocities

References

[1] Eckhard Krepper, Brahma Nanda Reddy Vanga, Alexandr Zaruba, Horst-Michael Prasser, and Martin A Lopez de Bertodano. Experimental and numerical studies of void fraction distribution in rectangular bubble columns. *Nuclear engineering and design*, 237(4):399–408, 2007.

[2] ANSYS Fluent. Fluent 2021 R2 user's guide. *Ansys Fluent Inc*, 2021.

[3] Florian Menter and Thomas Esch. Elements of industrial heat transfer predictions. *16th Brazilian Congress of Mechanical Engineering (COBEM)*, 20:117–127, 2001.

A. Appendix A: fvModels source code for degassing boundary conditions

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  3.0.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      fvOptions;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

degassingMassSource
{
    type            coded;
    active          yes;
    selectionMode   all;
    field       thermo:rho.air;

    name            degassingMassSource;
    patches         (outlet);
    rhoName         thermo:rho.air;
    alphaName       alpha.air;

    codeInclude
    #{
         #include "fvm.H"
    #};

    codeAddRhoSup
    #{

       Info << "**codeAddSup**" << endl;


         // Get the names of the patches on which to apply the degassing forces
         DynamicList<word, 1, 0> patches;
         coeffs().lookup("patches") >> patches;

         // Get the required fields
         const word rhoName = coeffs().lookup("rhoName");
         const volScalarField& rhoAir = mesh().lookupObject<volScalarField>(rhoName);
         const word alphaName = coeffs().lookup("alphaName");
         const volScalarField& alphaAir = mesh().lookupObject<volScalarField>(alphaName);

         // Get the timestep
         const scalar deltaT = mesh().time().deltaT().value();

         // Create degassing mass source coefficient and initialize to zero
         volScalarField degassingMassSourceCoeff
         (
```

```
            IOobject
            (
                "degassingMassSourceCoeff",
                mesh().time().timeName(),
                mesh(),
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            mesh(),
            dimensionedScalar("degassingMassSourceCoeff", dimless/dimTime, 0.0)
        );

        // Compute the degassing mass source coefficient for each cell adjacent to the
    selected patches
        forAll(patches, iPatch)
        {
            // Get the boundary patch
            const fvPatch& patch = mesh().boundary()[patches[iPatch]];

            // Loop through each boundary face and compute degassing force coefficient in
    adjacent cell
            forAll(patch, iFace)
            {
                label iCell = patch.faceCells()[iFace];
                degassingMassSourceCoeff[iCell] = -alphaAir[iCell]/deltaT;
            }
        }

        // Add the degassing force term
        eqn += fvm::Sp(degassingMassSourceCoeff, rhoAir);
    #};

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

}


degassingForce
{
    type            coded;
    active          yes;
    selectionMode   all;
    field        U.air;
    name            degassingForce;
    patches         (outlet);
    rhoName         thermo:rho.air;
    alphaName       alpha.air;
    UName           U.air;


    codeInclude
    #{
        #include "fvm.H"
    #};

    codeAddRhoSup
    #{
```

```
        // Get the names of the patches on which to apply the degassing forces
        DynamicList<word, 1, 0> patches;
        coeffs().lookup("patches") >> patches;

        // Get the required fields
        const word rhoName = coeffs().lookup("rhoName");
        const volScalarField& rhoAir = mesh().lookupObject<volScalarField>(rhoName);
        const word alphaName = coeffs().lookup("alphaName");
        const volScalarField& alphaAir = mesh().lookupObject<volScalarField>(alphaName);
        const word UName = coeffs().lookup("UName");
        const volVectorField& UAir = mesh().lookupObject<volVectorField>(UName);

        // Get the timestep
        const scalar deltaT = mesh().time().deltaT().value();

        // Create degassing force coefficient and initialize to zero
        volScalarField degassingForceCoeff
        (
            IOobject
            (
                "degassingForceCoeff",
                mesh().time().timeName(),
                mesh(),
                IOobject::NO_READ,
                IOobject::AUTO_WRITE
            ),
            mesh(),
            dimensionedScalar("degassingForceCoeff", dimDensity/dimTime, 0.0)
        );

        // Compute the degassing force coefficient for each cell adjacent to the selected
    patches
        forAll(patches, iPatch)
        {
            // Get the boundary patch
            const fvPatch& patch = mesh().boundary()[patches[iPatch]];

            // Loop through each boundary face and compute degassing force coefficient in
    adjacent cell
            forAll(patch, iFace)
            {
                label iCell = patch.faceCells()[iFace];
                degassingForceCoeff[iCell] = -rhoAir[iCell]*alphaAir[iCell]/deltaT;
            }
        }

        // Add the degassing force term
        eqn += fvm::Sp(degassingForceCoeff, UAir);
    #};

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


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

Listing 1: fvModels source code