Numerical Simulation of an Industrial Problem - 2

von Karman Institute for Fluid Dynamics

Free-falling cylinder in water

Supervisors:
Maria Faruoli, Matilde Fiore

Author:
Lorenzo Vallisa

2020-2021

Contents

List of Figures

List of Symbols

δ_{ij} Kronecker's delta
m_b Rigid body mass
u_b Rigid body vectorial velocity
ω_b Rigid body angular velocity
I_b Moment of inertia of rigid body
ρ_f Fluid density
$\boldsymbol{\tau}$ Stress tensor
V_b Rigid body volume
ρ_b Density of rigid body

Chapter 1

Introduction

In this report a numerical method is proposed to reproduce the dynamics of a free-falling cylinder into water. A mesh study is carried out to test the performances of the method and different turbulence models are tried out to quantify the influence of turbulence on the final result. Obtained numerical data are eventually compared with the results of experimental work for the final validation step.

1.1 Experimental Work and Setup

The article [9] we use as reference for the experimental data reports the results in which cylindrical objects of various weight and size were dropped into a pool filled with water. Their landing location and underwater trajectory over the course of their descent were observed and measured. The visual observation of trajectories was sketched and also noted in a descriptive way. These experiments were performed in a 2.5m (nominal) deep pool with the assistance of a diver. Nine cylindrical models were produced and dropped from a fixed height (to achieve a consistent initial velocity) and entry angle. Models were retrieved after all nine of them were dropped and results recorded. The most important variables to be tuned for each experiment were:

- Inclination of object at the moment of drop
- Relative position of centre of buoyancy and centre of gravity
- Relative size of falling cylinder

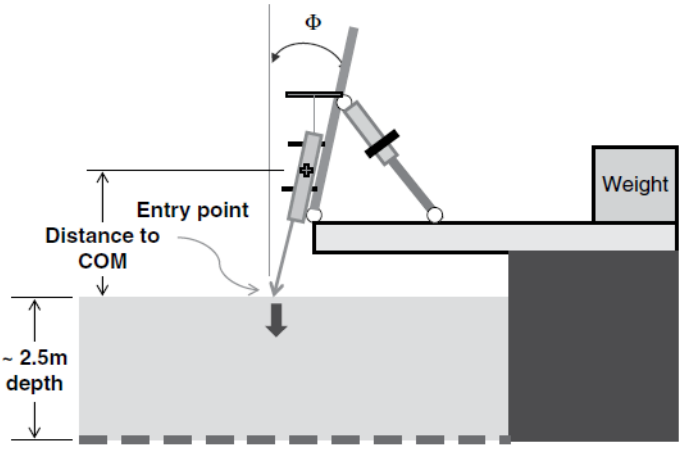


Figure 1.1: Experimental Setup

1.2 Physics Involved

The problem at stack is a multi-phase problem, in which a solid object crosses first an air zone, landing on a water surface of a pool and eventually diving into it. Lagrangian dynamics of the single object has to be reproduced, that is equation of motion for a rigid body is obtained from conservation of linear and angular momentum:

$$m_b \frac{d\boldsymbol{u}_b}{dt} = \rho_f \int_{\Gamma_p} \boldsymbol{\tau} \cdot \boldsymbol{n} ds + V_b(\rho_b - \rho_f)\boldsymbol{g} \tag{1.1}$$

$$I_b \frac{d\boldsymbol{\omega}_b}{dt} = \rho_f \int_{\Gamma_p} \boldsymbol{r} \times (\boldsymbol{\tau} \cdot \boldsymbol{n}) ds \tag{1.2}$$

No collisional phenomena with other objects are foreseen. For what concerns the fluid, the Navier-Stokes set of equations for incompressible isothermal

fluid is solved:

$$\boldsymbol{\nabla} \cdot \mathbf{u} = 0 \tag{1.3}$$

$$\rho \left[\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \boldsymbol{\nabla})\mathbf{u} \right] = -\boldsymbol{\nabla} p + \mu \nabla^2 \mathbf{u} + \rho \mathbf{g} \tag{1.4}$$

1.3 Aim of the Project

The aim of this project is to reproduce the test case mentioned in section 1.1. A full validation of the experimental results obtained by [9] won't be anyway possible: the number of simulations to be done to produce a sufficient amount of data to build a probability distribution (see Fig.1.2) is too high to be fully processed within the framework of this project. We will limit hence the analysis to reproduce a single experiment in the most accurate possible way.

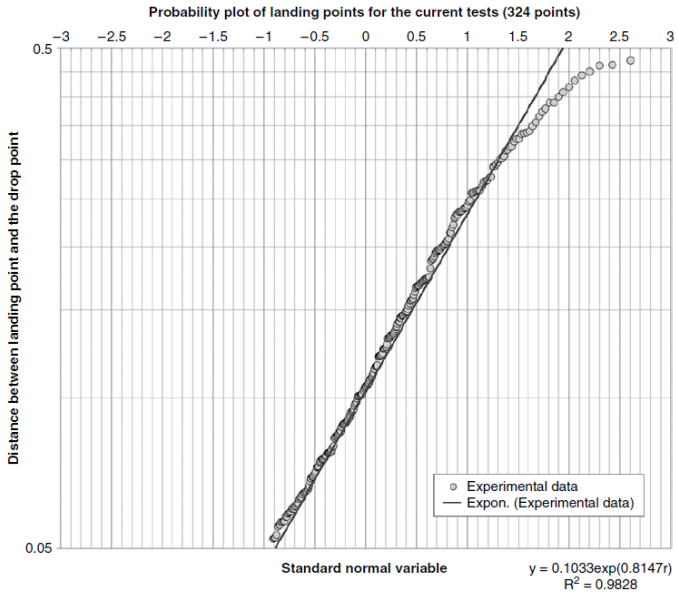


Figure 1.2: Linear Least Square Regression for Experimental Data

Chapter 2

Numerical Models and Settings

In this chapter a thorough analysis is carried out to investigate the CFD solver used for the fluid part, the mesh structure, as well as the mesh-solver used for interfacing the object with the different phases. Further initial and boundary conditions, different turbulence models and geometrical modeling will be discussed. The software used for this project framework will be *OpenFOAM* (v2012).

2.1 CFD Solver

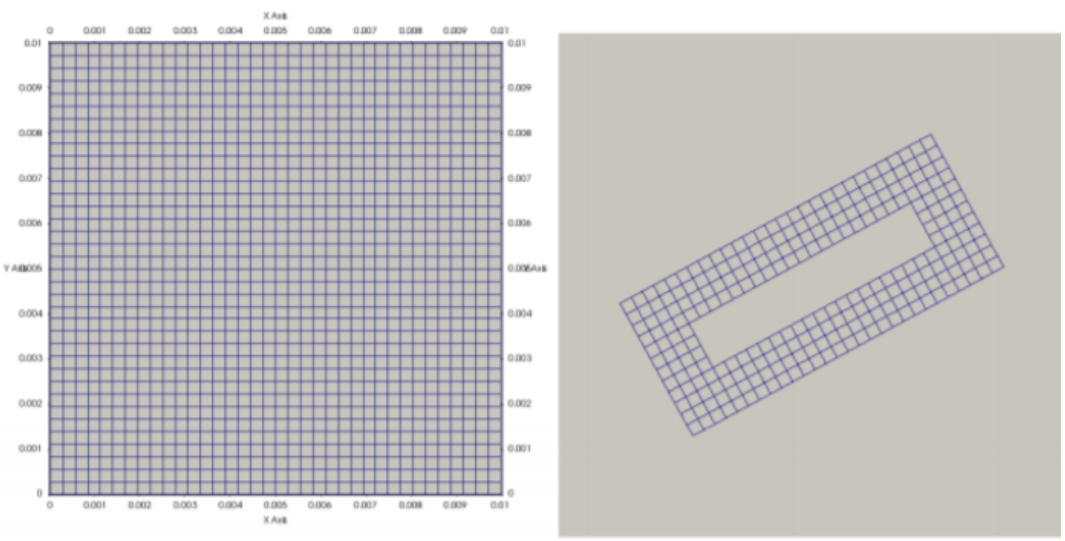
The CFD solver used is the OpenFOAM *overInterDyMFoam* algorithm. It is a solver for two incompressible, isothermal immiscible fluids using a VOF (volume of fluid) phase-fraction based interface capturing approach, with optimal mesh motion and mesh topology changes including adaptive re-meshing ([5]). Mesh solver features will be discussed into details in section 2.2.1. In the *overInterDyMFoam* algorithm fluid equations are solved using OpenFOAM *PIMPLE* algorithm [6]: equations for each variable characterizing the system (velocity,pressure and variables characterizing turbulence) is solved sequentially and the solution of the preceding equations is inserted in the subsequent equation. The non-linearity appearing in the momentum equation (the face flux ϕ which is a function of the velocity) is resolved by comput-

ing it from the velocity and pressure values of the preceding iteration. The dependence from the pressure is introduced to avoid a decoupling between the momentum and pressure equations and hence the appearance of high frequency oscillation in the solution. The first equation to be solved is the momentum equation. It delivers a velocity field which is in general not divergence free, i.e. it does not satisfy the continuity equation. After that the momentum and the continuity equations are used to construct an equation for the pressure. The aim is to obtain a pressure field, which, if inserted in the momentum equation, delivers a divergence free velocity field. After correcting the velocity field, the equations for turbulence are solved [8]. The outer correctors are the iterations, and once converged will move on to the next time step until the solution is complete [6]. The number of outer correctors of the PIMPLE algorithm, as above mentioned, defines how many outer iterations to perform—that is, how many times the system of equations are performed before it is forced to move onto the next time step, regardless of whether that time step has converged or not.

2.2 Mesh

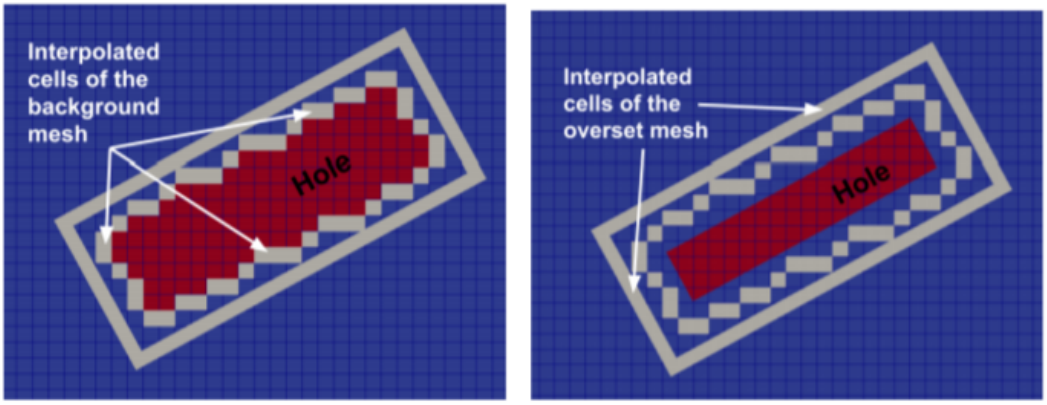
2.2.1 Solver

The identification of the mesh solver for this project is indeed the most delicate part. A first choice was the *Dynamic Mesh* solver in which, at each time step, a new mesh structure is provided to follow correctly the dynamics of the object. The main issue with this choice was that the solver was not able to reproduce the force interaction happening on the solid surface when transiting form a phase to another one: simulation was in fact diverging each time the object impacted on the water surface. The second algorithm proposed was the *Overset Grid* (OSG) solver , on which the *overInterDyMFoam* solver mentioned in section 2.1 is based. In the OSG ([1]), two grids (background and bodyfitted) are defined, which may arbitrarily overlay each other. The different grids are internally static, thereby retaining their original structure and quality, but are allowed to move relative to each other. In order to pass information between the different grids, interpolation has to be performed. Governing partial differential equations are solved on both the background



(a) Background mesh

(b) Overset mesh



(a) Hole (red cells) and interpolated cells (white cells) defined for background mesh

(b) Hole (red cells) and interpolated cells (white cells) defined for overset mesh

Figure 2.1: Cell differentiation in overset method: the overset mesh body-fits the object and overlaps with the background one

mesh and the overset mesh. Within the background domain, the elements corresponding to the interior of the overset domain are marked as holes and removed from the computational domain. The values on the boundary of the overset domain are then interpolated to the background mesh. Similar pro-

cedure is realized for the overset domain. As a result, in each timestep, each cell of the background and overset mesh is marked as one of the following types:

- *calculated*, for which equations are solved
- *interpolated*,values in these cells are computed by interpolation from the nearest elements of the second domain
- *holes*, no solution is computed here

In Fig.2.1 background and body-fitted overset meshes are shown togheter with the different kind of type-cells for each of the two ([7]).

2.2.2 Structure

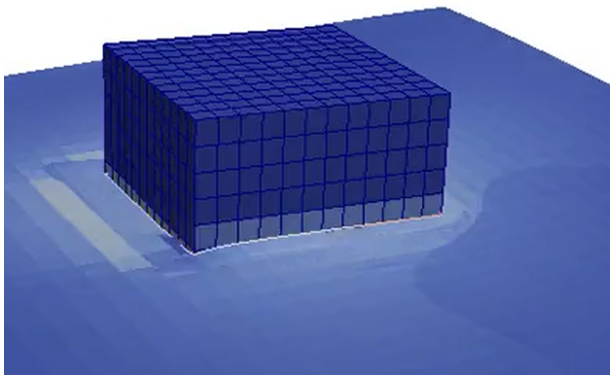


Figure 2.2: Tutorial test-case: floating object

The very first meshing strategy adopted is the one of starting with the settings of a tutorial test case which the most resembles the physics we are dealing with: a floating object ([2]) test case was hence chosen as a starting point (see Fig 2.2). From Fig.2.3 it is possible to see the different domains applied to the floating object test cases, in which the object is the smallest of the three outline, whereas in Fig.2.4 the division of the two main phases

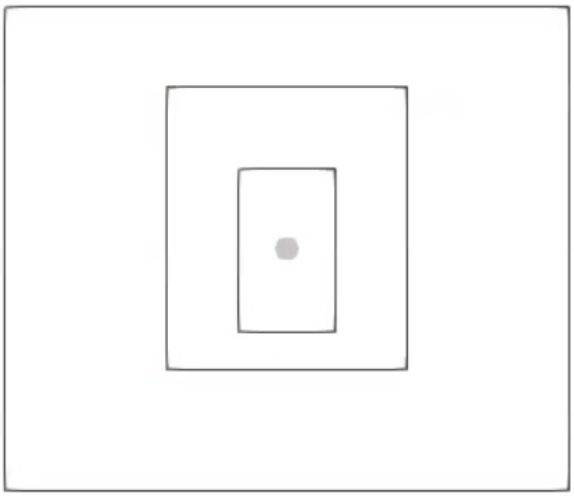


Figure 2.3: 2D slice (z-x plane) of domain for tutorial case: outline of the floating object surrounded by overset grid and external domain

(water and air) on domain is shown. The features of the above mentioned starting test-case are then modified to be scaled according to our problem: the ratio between the overset grid size, and the size of the object are adjusted accordingly, whereas the external domain and the object shape and size are adopted to the features of the full-scale experimental problem. The pool is modeled hence as a cube, whose depth and base square side length is 2.5 [m].

Atmosphere is more of a cuboid shaped geometry, since the height is taken

[m,-,mm]	**Size: (x,y,z)**	**Cells n°:(x,y,z)**	**Cells size:(x,y,z)**
Pool - Atmosphere	(2.5,2.5,2.5 - 1.0)	(80,80,90)	(31,31,35)
Overset	(0.8,0.8,0.8)	(75,75,75)	(11,11,11)

Table 2.1: Initial Mesh Structure Settings (Test 1 Tab. 3.1)

as 1.0 [m]. Always according to [9], the falling cylinder has to be in a $20:1$ ratio w.r.t. the depth of the pool, hence starting length of the cylinder is chosen to be 12 [cm], with radius 4 [cm] (See Fig.2.5). As already outlined in the choice of the mesh solver in section 2.2.1, we will have to define a region for the *overset* grid to surround the falling object, and we will chose it so that it will be restricted to a single phase initially (hence only in air at the

Figure 2.4: 2D slice of domain for tutorial case: blue is air phase, red is water

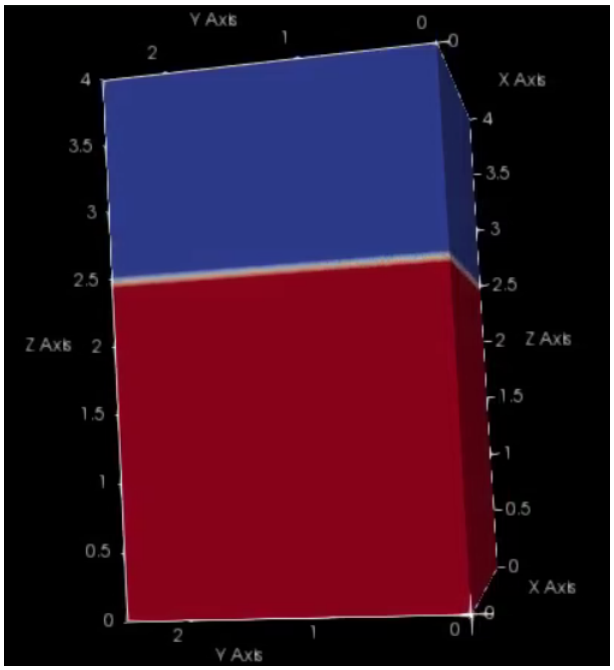


Figure 2.5: Current test case domain with frame of reference and axis

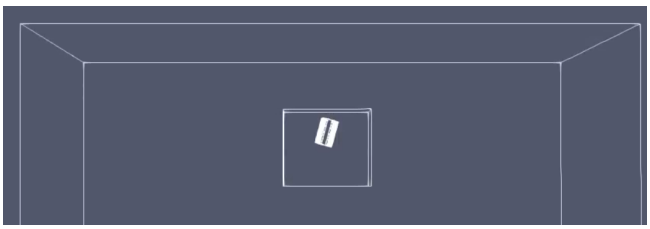


Figure 2.6: Slice of the z-x plane

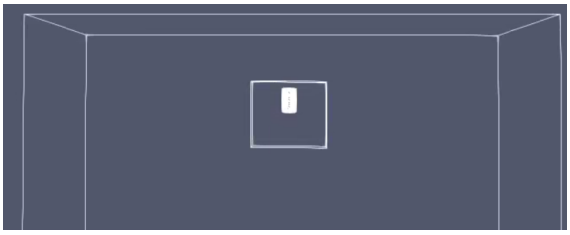


Figure 2.7: Slice of the z-y plane

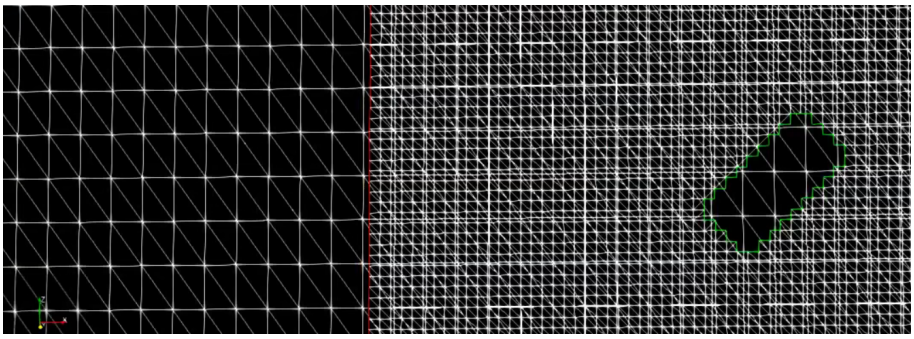


Figure 2.8: Zoom on 2D slice (z-x plane) of full mesh on current test-case: overset grid is more refined than external grid

beginning), and to be much more refined than the external outer mesh. Features of this preliminary mesh design are recapped in Tab. 2.1 and showed in Fig.2.8. The main reasons behind this choice are the following:

- Refining part of the external mesh where nothing is really happening might be a waste of computational energies
- Since object size is smaller than the one of floating object of the original test case, an higher refinement in the overset grid can be a good strategy to better capture the features of the falling cylinder

We will see eventually in section 3.1 that different strategies might be adopted, according to a thorough parametric study, to reduce computational cost and still keep a good enough final solution.

2.3 Initial and Boundary Conditions

In the following section the initial and boundary conditions chosen for our problem will be listed and justified. Within OpenFOAM initial and boundary conditions are specified within the **background/0** folder: next to the voice *internalField* the former apply, whereas in the *boundaryField* brackets field the latter one can be specified for each of the domain part:

- *stationaryWalls* (sW), the walls of the external mesh domain
- *atmosphere* (a), boundary between the two phases' domains
- *floatingObject* (fO), the surface of the object interacting with the surrounding phase domain
- *oversetPatch* (oP), the boundary between the external mesh and the overset grid

2.3.1 pointDisplacement (pD)

Field variable responsible to account for solving and recording movement of each mesh point. *fixedValue* (fV) means that the mesh displacement happening on that precised boundary is fixed: set to 0 for both atmosphere and external domain walls. *calculated* (c) is enforced on the *floatingObject* surface: throughout the simulation the mesh displacement happening at this interface is updated. On the *oversetPatch* a *zeroGradient* (zG) condition is enforced, meaning that on that boundary the mesh deformation should be exactly the same for the two grids (background and body-fitted) sharing that interface.

2.3.2 p_rgh

Variable p_{rgh} in OpenFOAM is the pure pressure variable, hence without the gravity adjustment term, which is included when computing the full pressure

term p:

$$p = p_{rgh} + \rho g h \tag{2.1}$$

fixedFluxPressure (fFP) is enforced on *stationaryWalls*,*floatingObject* and *oversetPatch*: the pressure gradient on these boundary is set to the provided value such that the flux on the boundary is that specified by the velocity boundary condition (**U**=0).

2.3.3 U

For the boundary part in which water is in contact with air a *pressureInletOutletVelocity* (pIOV) type of BC is set: a *zeroGradient* outflow is applied and the velocity is obtained from the patch-face normal component of the internal-cell value. This to respect the interaction between a much denser phase (water) on the bottom interfacing a less dense one (air) at the top. The condition applied for the cylinder is the *movingWallVelocity* (mWV), typical condition used for moving bodies.

2.3.4 Recap Initial Condition (IC) and Boundary Condition (BC)

For a full documented literature on all possible boundary and initial conditions that can be set within OpenFOAM, please check [4]. In Tab 2.2 a recap of the different values chosen for the four main boundaries listed at the beginning of this section, as well as initial conditions.

-	**pD**	$\mathbf{p}_{rgh}$	**U**
sW	fV (0)	fFP	fV (0)
a	fV (0)	fv (0)	pIOV (0)
fO	c	fFP	mWV
oP	zG	fFP	zG
IC	0	0	0

Table 2.2: Boundary and Initial Condition

2.4 Turbulence Models

The first issue to tackle is whether actually using turbulence models or not. One way to estimate whether the use of a turbulence model is actually necessary is through a rough computation of the Reynolds number. Considering a maximum vertical velocity ([9]) of the object $w = 2.2[m/s]$, diameter of the cylinder $d = 0.08[m]$ and density and dynamic viscosity of water, the Reynolds number is as follows:

$$Re = \frac{dw\rho}{\eta} = 160000 \tag{2.2}$$

which is above the critical Reynolds number for flow past cylinder ([3]), and hence sufficiently high to introduce the use of a turbulence model. Two main models were tested within this projects, and both of them belonging to the family of RANS models. The main interest of this study is to be able to correctly reproduce the dynamics of the object, therefore a thorough and exhaustive representation of fluid structure is not needed. This is why we believe a RANS turbulence model can be used within this project framework. The main two RANS models that are taken into account for this work are the k-ε and the k-ω ones. Since the object is actually subject to a free fall, both in air and in water, it is believed that no strongly adverse pressure gradient would be experienced on the surface of the object interacting with the surrounding fluid, and hence a k-ε model will be sufficient. Here below the initial and boundary condition for the two turbulence models mentioned above:

Listing 2.1: epsilon initial condition

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

internalField uniform 0.1;

boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    stationaryWalls
    {
        type            epsilonWallFunction;
```

```
        value           uniform 0.1;
    }

    atmosphere
    {
        type            inletOutlet;
        inletValue      uniform 0.1;
        value           uniform 0.1;
    }

    floatingObject
    {
        type            epsilonWallFunction;
        value           uniform 0.1;
    }
}
```

Analogous initial conditions for the ω variables are proposed here under, considering a value for $C_\mu = 0.09$ in:

$$\omega = \frac{\varepsilon}{C_\mu k} \tag{2.3}$$

Listing 2.2: omega initial condition

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

internalField uniform 11;

boundaryField
{
    #includeEtc "caseDicts/setConstraintTypes"

    stationaryWalls
    {
        type            omegaWallFunction;
        value           uniform 11;
    }

    atmosphere
```

```
    {
        type            inletOutlet;
        inletValue      uniform 11;
        value           uniform 11;
    }

    floatingObject
    {
        type            omegaWallFunction;
        value           uniform 11;
    }
}
```

Results of which of the two turbulence models can suit our problem at best will be presented in the section 3.3.

2.5 Geometrical Modeling

The object default initialized in the original OpenFOAM test-case for floating object ([2]) is a cuboid, hence some modification have to be brought about to eventually simulate a cylindrical shaped object: two are the ways. A more cumbersome one is to create an object using a CAD software, export the *.stl* file and then use *snappyHexMesh* to build the mesh around the object. Within this project framework the strategy adopted is the one to use the feature *cylinderToCell* in *topoSetDict* (see Fig.2.9), which allows in a much easier way to obtain a cylindrical shaped object: since the aim of this project is to verify the validity of the method, we thought the *topoSetDict* approach would be a sufficiently valid approach.

2.5.1 Cylindrical Shape

The way we first exploit to try and simulate a cylindrical shape is to work with the *topoSetDict* file (see Fig.2.9), which essentially works by considering the edges of the object, defined as a box of round shaped cells along the z-

```
actions
(
        {
        name    c0;
        type    cellSet;
        action  new;
        source  cylinderToCell;
        p1      (1.25 1.25 3.44);
        p2      (1.25 1.25 3.56);
        radius  0.04;
        }
    {
        name    c0;
        type    cellSet;
        action  invert;
    }
);
```

Figure 2.9: topoSetDict settings for cylindrical shape object

axis only. By refining the mesh for the overset grid, it is indeed possible to improve the shape of the cylinder as shown in Fig. 2.11.

2.5.2 Dynamical Features

The file in OpenFOAM that deals with this object's features is *dynamicMeshDict* file responsible for the dynamic of the mesh, more precisely the coefficients to be tuned for the interaction of the object with the mesh are those listed below the *sixDoFRigidBodyMotionCoeffs*.

```
motionSolverLibs  (sixDoFRigidBodyMotion);

dynamicFvMesh     dynamicOversetFvMesh;

solver         sixDoFRigidBodyMotion;

sixDoFRigidBodyMotionCoeffs
{
    patches       (floatingObject);
```

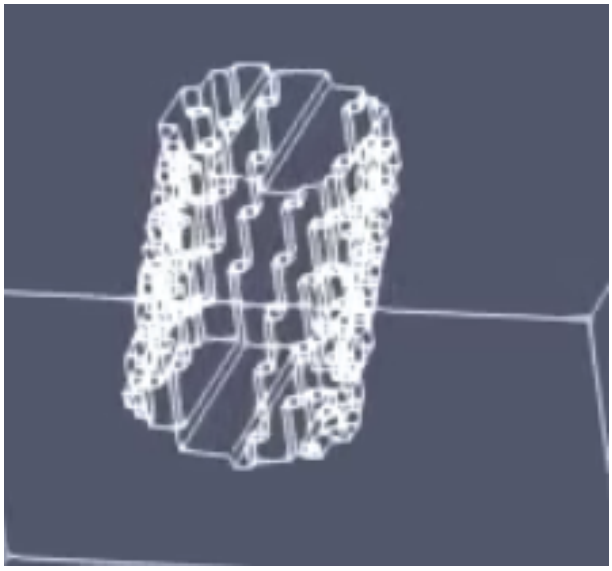


Figure 2.10: Coarse overset cylinder

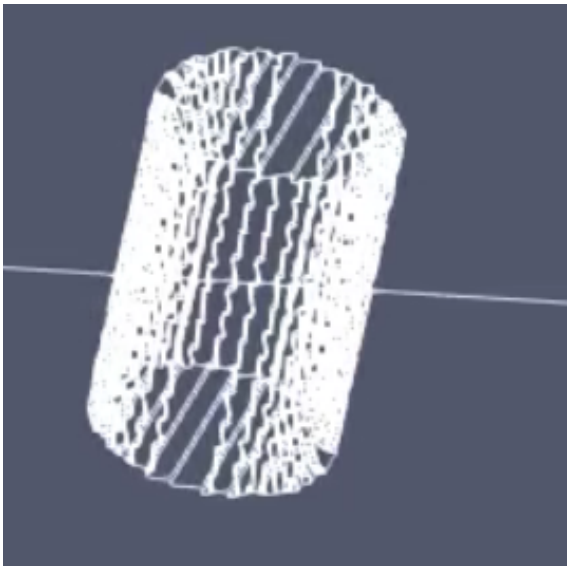


Figure 2.11: Refined overset cylinder

```
innerDistance  100.0;
outerDistance  101.0;
// Cylinder dimensions
R           0.04;
Lz             0.12;
centreOfMass   (1.25 1.25 3.5);
// Density of the solid
rhoSolid       1300;
// Cylinder mass
mass           #eval{ $rhoSolid*$R*$R*3.14*$Lz };

// Cylinder moment of inertia about the centre of mass
momentOfInertia #codeStream
{
    codeInclude
    #{
        #include "diagTensor.H"
    #};

    code
    #{
```

```
            scalar sqrR = sqr($R);
            scalar sqrLz = sqr($Lz);
            os  <<
                $mass
               *diagTensor(3*sqrR + sqrLz,3*sqrR + sqrLz,
                   6*sqrR)/12.0;
        #};
    };
    report        on;
    accelerationRelaxation 0.1;
    accelerationDamping  1.0;
    solver
    {
        type Newmark;
    }
    constraints
    {
    }
}
```

2.5.2.1 Center of Mass and Orientation

The position of the center of mass is updated according to the angle between the vertical axis and the main symmetry axis of the cylinder, and according to the length of the cylinder. In an analogous fashion also the features of the *topoSetDict* have to be adjusted to account for the inclination of the cylinder around the y-axis. To comply with this changing a Python code (see Fig.2.12)has been written, that for each inclination angle and cylinder length, recomputes the coordinates of the center of mass and of the boxes to be allocated within the topoSetDict. An example of tilted geometry is shown in Fig.2.13

```python
@author: Lorenzo Vallisa
"""
import numpy as np
import io
import matplotlib.pyplot as plt
import os
import math

#INPUT for topoSetDict settings
L = 0.12
angle_deg = 45

#OBTAINED QUANTITIES
angle= angle_deg/180 * np.pi
a = L/2;

x_min = 1.25 - a * math.sin(angle)
z_min = 3.5 - a*math.cos(angle)

x_max = 1.25 + a*math.sin(angle)
z_max = 3.5 + a*math.cos(angle)

print('topoSetDict features')
print('(',x_min,1.25,z_min,') ,','(',x_max,1.25,z_max,')')

#INPUT for COM settings
b = 1 #+1 above - 0 coindice - -1 below
beta = L/3


#OBTAINED QUANTITIES
x = 1.25 + b*beta*math.sin(angle)
z = 3.5 + b*beta*math.cos(angle)

print('COM')
print('(',x,1.25,z,')')

```

Figure 2.12: Python script to adjust features for setTopoDict

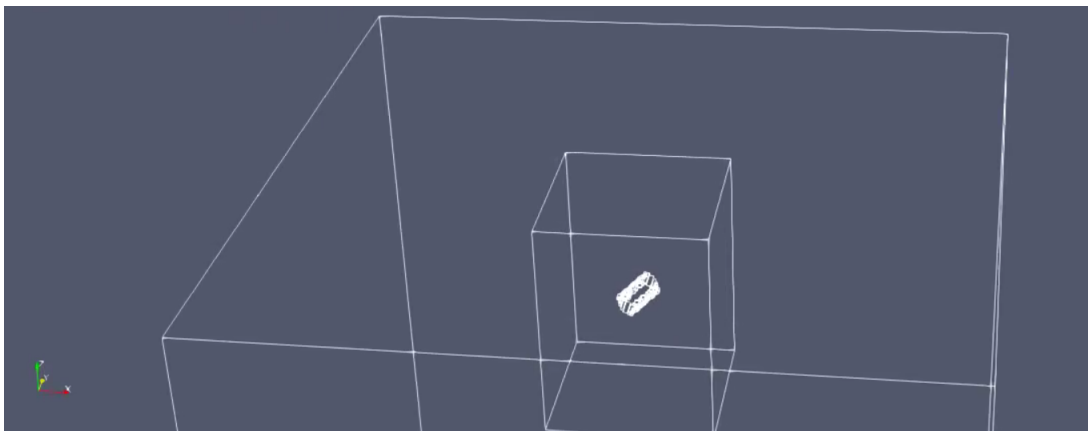


Figure 2.13: Tilted Cylinder by 45°

2.5.2.2 Moment of Inertia

The *moment of inertia* has been modified according to the equation for the cylinder

$$I = \begin{bmatrix} \frac{1}{12}m\left(3r^2 + h^2\right) & 0 & 0 \\ 0 & \frac{1}{12}m\left(3r^2 + h^2\right) & 0 \\ 0 & 0 & \frac{1}{2}mr^2 \end{bmatrix} \tag{2.4}$$

Chapter 3

Results

3.1 Final Numerical Settings: parametric analysis to reduce CPU timings

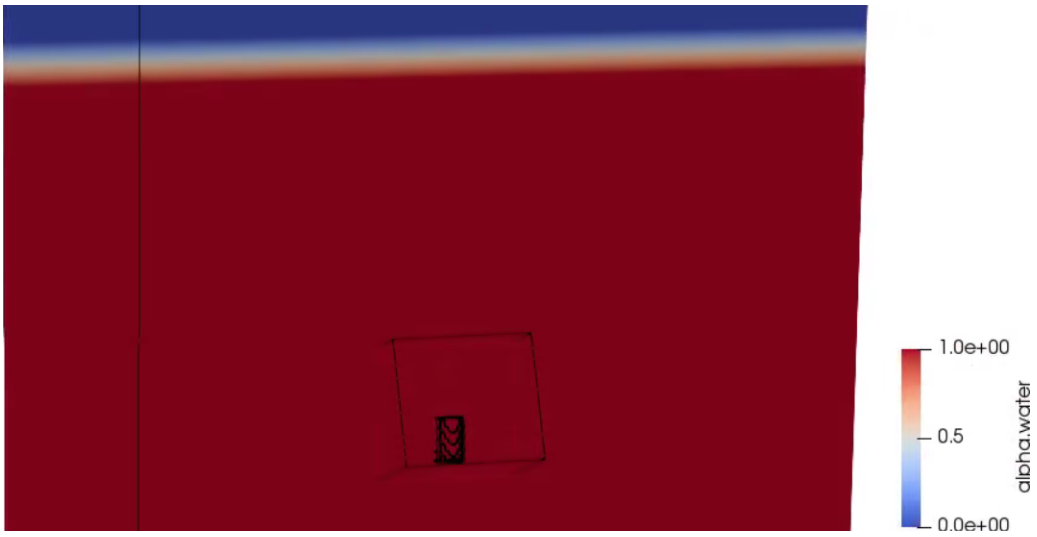


Figure 3.1: Test 6 (from Tab.3.1) outline of cylinder with slice of phase surface along z-x plane

In this section a parametric study on the computational efficiency of our settings is presented: different strategies are taken into account and each improving step is hence justified. The starting point, as already outlined in section 2.2.2, was to run a simulation with numerical settings as close as possible to the ones of the tutorial case, but having an overset grid initially crossing two phases generated an uncontrolled growth of the cylinder velocity each time the falling object was about to impact the water surface, generating thus a diverging simulation. In Tab. 3.1 a list of the main simulation settings that were tested is presented. The legend of symbols used in Tab. 3.1 is:

Test	1	2	3	4	5	6
$\frac{Overset}{Object}^{***}$	8	4	2	2	2	**4**
MPI	N	Y	Y	Y	Y	Y
Overset Cells	75	55	55	45	45	45
Ext. Mesh Cells	80**	60**	60**	50**	50**	(10,10,60)
Relaxation Factor	0.1	0.1	0.1	0.1	0.5	0.8
MAX CFL	2	2	2	2	3	3
Acc. Relax	0.1	0.1	0.1	0.1	0.3	0.3
Acc. Damp	1.0	1.0	1.0	1.0	0.95	0.95
CPU Time	∞	**25h***	**25h***	**20h***	**15h**	**20m**

Table 3.1: Parametric Study Table: settings are those in first line of Tab 3.6

- symbol *** refers to the $\frac{Overset}{Object}$ ratio, that is the ratio between the dominant size in the overset method and the dominant size of the solid object
- symbol ** means that the amount of cell written is the one on the x-y plane,whereas on the z-axis, ten more cells are provided
- symbol * means the CPU total time was estimated by counting how much time is it needed to run a just a part of the simulation

The very first step is to reduce the amount of cells for each grid involved, since, as already justified in section 2.2.2, not much refinement is needed in part of the domain where nothing is happening. Since no strong improvement is appreciated, a fine tuning of others numerical settings such as relaxation

factors, CFL, acceleration relaxation[1] and acceleration damping[2] is introduced. The key to obtain a reliable result in a short amount of time (most right column of Tab. 3.1) is to allows for lower acceleration relaxation and damping, increasing the relaxation factor up to 0.8 (maximum value for the simualtion to remain stable) and the CFL up to 3.0 (PIMPLE algorithm allows stable simulation to be run with CFL slightly higher than 1 [6]). Increasing eventually the $\frac{Overset}{Object}$ ratio, allows for a much smoother transition between the information on the fluid domain towards the solid object passing through the interpolating overset grid, resulting thus in the lowest CPU time among all the simulation run for this test-case. We can conclude that it is possible to run a simulation (column 6 in 3.1)

- full-scale
- cylindrical shaped with good resolution
- in little more than 20 minutes

3.2 Grid Refinement Analysis

Before proceeding to validate the test-case, it is essential to understand whether the numerical settings discussed and eventually chosen in section 3.1, are not only the one that give a quick result but also an accurate one. In order to check this result, a mesh sensitivity analysis has to be brought about and two main strategies are proposed. A very first strategy involves grid refinement through a combined domain size reduction and cells number increase, whereas a second mesh refinement strategy will be proposed in which the only parameter to be changed is the number of cells within the grid. A final important remark has to be done on how the discrepancy of two trajectories has been computed. Given two trajectory z_1 and z_2 the

[1] feature of the *sixDoFRigidBodyMotionCoeffs* that allows to relax the forces computed at the interface bewteen the solid and the fluid: 0.1 means maximum relaxation applied

[2] feature of the *sixDoFRigidBodyMotionCoeffs* that allows to introduce a damping on the forces computed at the interface bewteen the solid and the fluid: 1.0 means maximum damping applied

discrepancy parameter D has been estimated using the following formula:

$$D = \max_{x,y} \frac{|z_1(x,y) - z_2(x,y)|}{|min(z_1, z_2)|} \tag{3.1}$$

where $z(x,y)$ is the height (z axis) of the solid given its position on the x-y plane. Only value below 1.5 % will be accepted when saying that two trajectories can be assumed to be *close enough*.

3.2.1 First strategy: grid refinement combined with domain size reduction

The main reason why a grid refinement was proposed through a combined effect of domain size reduction and cells number increase was because of the results obtained in section 3.1: indeed the impact of the size of the overset grid on the final result will be tested through a mesh sensitivity analysis, trying to keep in this way the computational time for the simulation as low as possible. One essential thing to be noted is that refining the mesh does not imply simply adding more cells for the same grid type, since this mere process slows down the computation in a dramatic way (see for example results obtained in Tab. 3.1), especially when the object experiences unsteadiness in the water phase. One solution that is proposed is to obtain a refinement working with the combined scaling of overset grid size and mesh size: in this way way less cells are added, compromising with the fact that the overset grid surrounding the object is now also smaller. The trajectory of a simulation[3] with a coarse mesh is compared with the one of a simulation in which the mesh has been slightly refined and with another one in which the mesh has been strongly refined (for details see Tab.3.5). If two curves can be considered *close enough* ($D < 1.5\%$ from equation 3.1), then the mesh has basically reached convergence, and the result we obtain with the coarser of the two meshes can be considered accurate enough. The refinement proposed, both for internal mesh and overset grid mesh, is outlined in Tab.3.5. When working with *overset* method a higher level of grid refinement means not only a higher level of detail of the object's shape (see Fig.2.10 and Fig.2.11) but also a more accurate transmission of information between the object dynamics and the surrounding fluid domain. That is essentially why, priority was

[3] **Simulation 4** from Tab.3.6

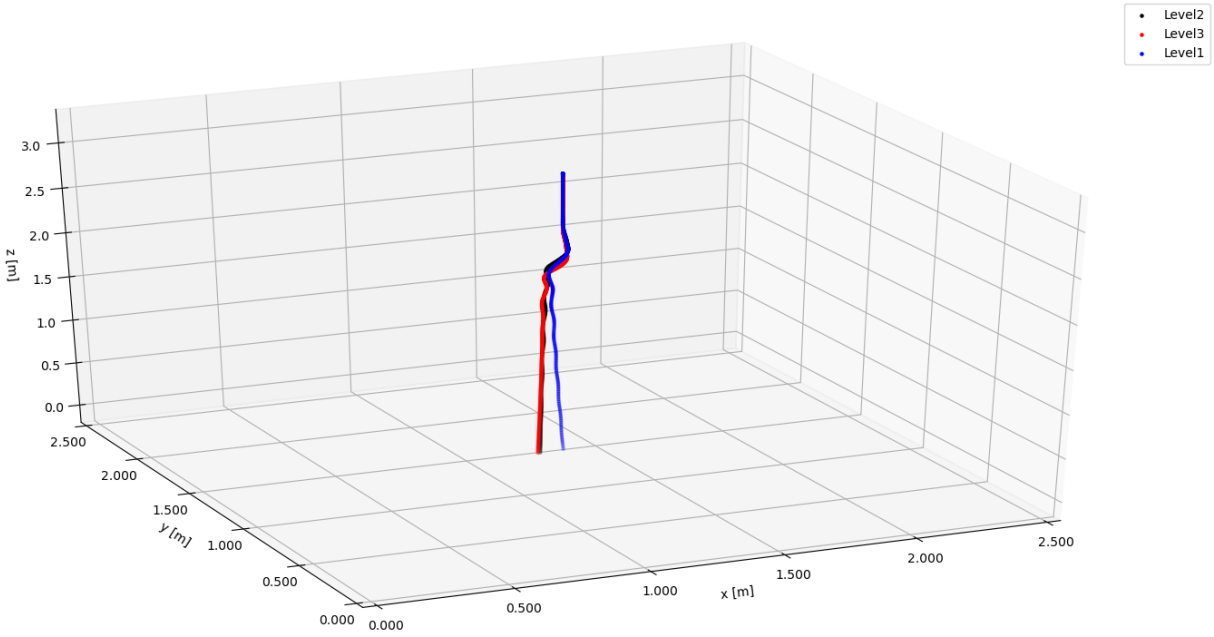


Figure 3.2: Trajectories comparison with refined mesh

(x-y,z)	Overset grid cell size [m]	External grid cell size [m]
Level 1	(0.0125,0.011)	(0.25,0.07)
Level 2	(0.0085,0.008)	(0.25,0.058)
Level 3	(0.00425,0.004)	(0.125,0.038)

Table 3.2: Level of refinement for each mesh

given to the refinement of the overset gird, whose cell's size has been halved in all directions. As it is shown in Fig.3.2, among the three trajectories obtained with different meshes, the two with the highest level of refinement are the ones resembling the most each others. It is not possible anyway to conclude that with the **Level 2** cell size values listed in Tab.3.5 the mesh is close to convergence[4] even though the results we are analysing are produced in a relatively short amount of time (compare with results in section 3.2.2). On Fig.3.3[5] and Fig.3.4 it is further possible to appreciate the grid refinement

[4]From 3.3 and 3.4 it is possible to see how the maximum relative error between trajectories of Level 2 and Level 3 is actually almost 2% (using D from equation 3.1)

[5]next to the Level number in parenthesis the CPU time needed to run the simulation up to that point

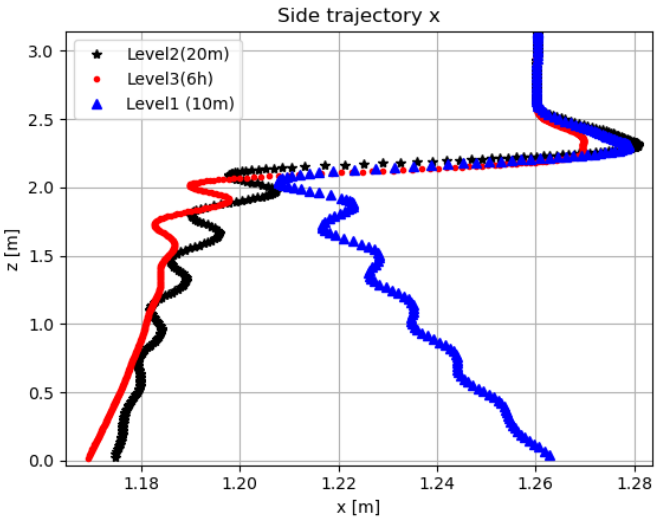


Figure 3.3: 1^{st} strategy: trajectory on x-z plane

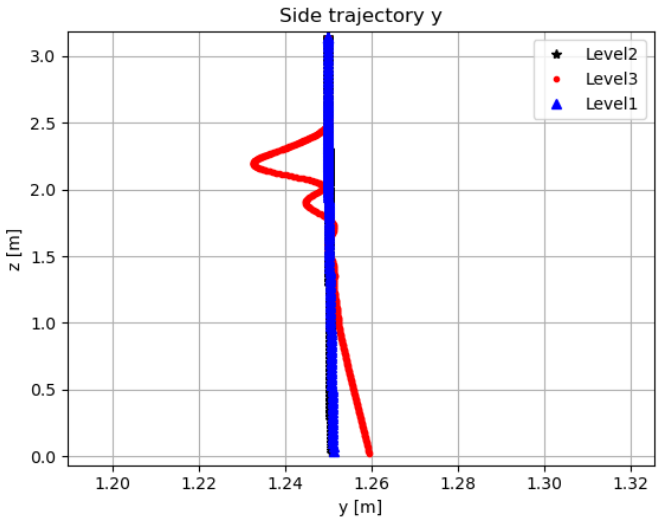


Figure 3.4: 1^{st} strategy: trajectory on y-z plane

influence on trajectory profile projected along both planes perpendicular to the x-y plane. The reason behind the fact that the trajectories differ a bit in certain points, may lay in the fact that the size of the overset grid domain changes: as already mentioned in section 2.2.2, the overset grid is responsible for the communication between the solid object and the internal mesh domain, hence by reducing its size, the interpolation procedure happens on a much smaller spatial scale and the information reaching the object from the fluid external zone might be slightly lagged (see Fig.3.3 and Fig.3.4). This is also the reason why a second refinement strategy has been proposed in the following section.

Levels (i,j)	$\mathbf{D}_{i,j}$
1-2	5 %
2-3	1.8 %

Table 3.3: Quantitative analysis of discrepancy between two trajectories using D parameter from equation 3.1 in first mesh refinement strategy

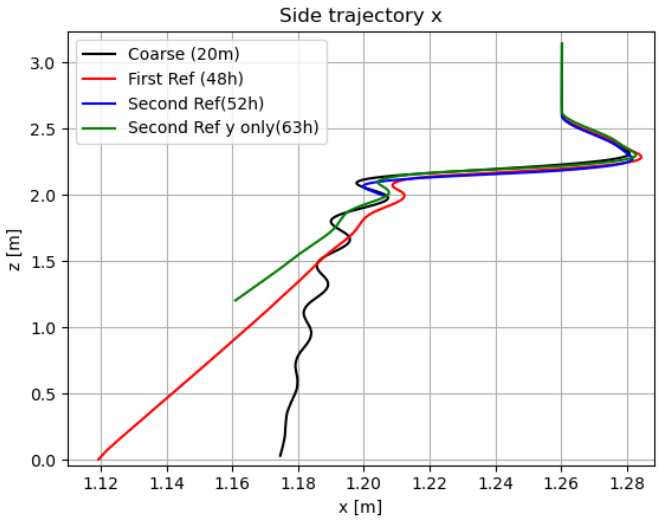


Figure 3.5: 2^{nd} strategy: trajectory on x-z plane

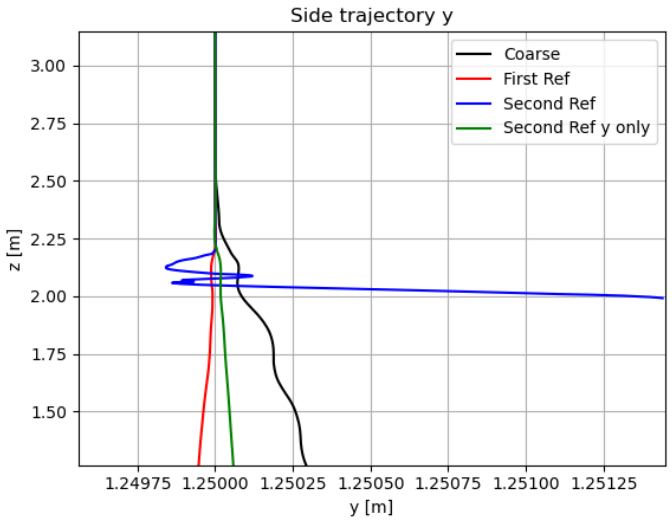


Figure 3.6: 2^{nd} strategy: trajectory on y-z plane

3.2.2 Second strategy: grid refinement through mere cells' number increase

Grid refinement can also be achieved by simply adding more cells within each domain, and in this specific case it is also very helpful to compare results with the ones obtained in section 3.2.1, to see whether they resemble or not. Being that not the case in fact, this would be a further proof that the size of the overset domain plays a key role in the accuracy of the desired result and the shape of the final trajectory. According to Fig. 3.5[6] and Fig. 3.6 the following things can be observed:

- Increasing the amount of cells only results in almost unbearable computational costs, since the interpolation procedure between the background and the body-fitted overset grid becomes very costly
- Results are much different w.r.t. those obtained by combining the cells number increase with a domain size reduction, meaning that, as

[6]next to the Level number in parenthesis the CPU time needed to run the simulation up to that point

already inferred from the conclusion pulled in section 3.2.1 the size of the domain of the overset do plays a key role

- Mesh convergence is still far: from a first level of refinement it is possible to see how the relative distance between the two trajectory (Coarse and First Ref) is around 4 % (using D from equation 3.1).

A final thing to be notice is that, by refining the mesh along the y axis only, oscillations observed in Fig.3.5 are somehow damped.

(x-y,z)	**Overset grid cell size [m]**	**External grid cell size [m]**
Coarse	(0.0085,0.008)	(0.25,0.06)
First Ref	(0.0072,0.0069)	(0.16,0.05)
Second Ref (y only)	(0.0072,0.0064)	(0.16,0.045)
Second Ref	(0.0061,0.006)	(0.125,0.04)

Table 3.4: Mesh refinement recap

(i,j)	$\mathbf{D}_{i,j}$
Coarse - First Ref	5.3 %
First Ref - Second Ref	4.8 %
First Ref - Second Ref (y)	> 2 %

Table 3.5: Quantitative analysis of discrepancy between two trajectories using D parameter from equation 3.1 in second mesh refinement strategy

3.2.3 Effect of external mesh

A very interesting study is proposed in the following section, mostly for the interest of future adventurers of the *overset* numerical method. Since it has already been shown as refinement procedure improves accuracy of the solution amplifying the computational costs beyond bearable limits, it would be interesting to see whether only one of the two grid can be refined leaving the other one untouched. Working indeed only with the overset grid might strongly simplify the work, but also the computational time to run simulations. Results are obtained comparing two simulations in which one has only the external mesh refinement increased by 20% with respect to the

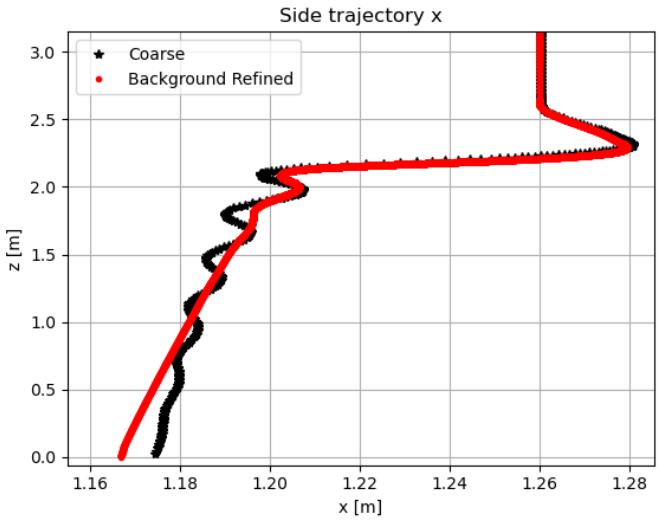


Figure 3.7: External mesh effect on x-z plane

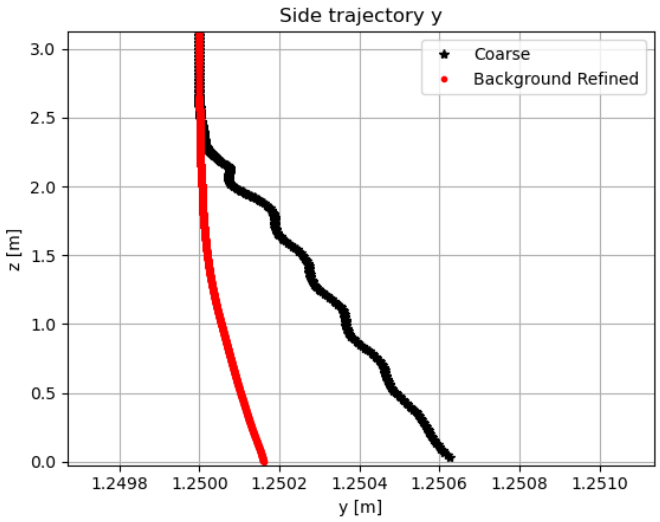


Figure 3.8: External mesh effect on y-z plane

other one. As it is possible to see from Fig.3.7 and 3.8, the discrepancy (D from equation 3.1) from the two trajectory is actually not small, up to almost 2%: hence it is not possible to conclude that the external mesh does not influence the accuracy of the result.

3.3 Choice of turbulence model

According to results obtained in the previous section a comparison between different turbulence models, introduced in section 2.4, is proposed to check whether using $k-\omega$ RANS model can result in major difference for the trajectory profile with respect to a $k-\varepsilon$ RANS model. Results from testing the two different turbulence models on simulation 10 from Tab. 3.6, are shown in Fig.3.9 and 3.10: due to the very low discrepancy in the trajectory profiles (less than 0.1 % according to equation 3.1), the usage of one of the two models is therefore interchangeable. For this reason in all the simulations presented in this report the $k-\varepsilon$ RANS model is used.

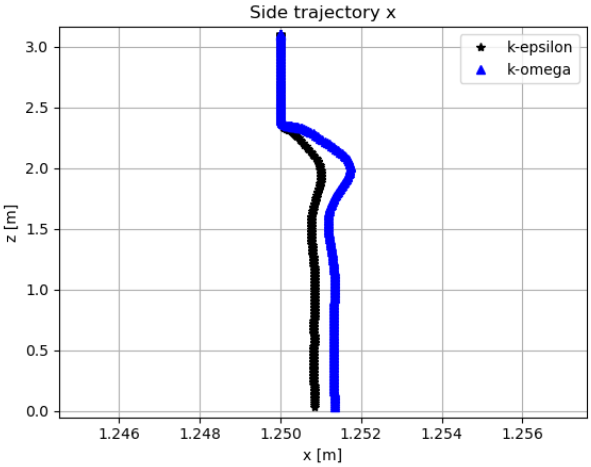


Figure 3.9: Turbulence comparison on x-z plane

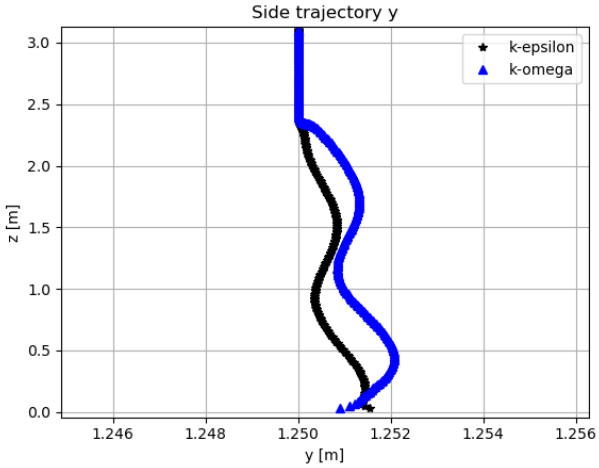


Figure 3.10: Turbulence comparison on y-z plane

3.4 Validation Test-Case

3.4.1 Statistical Model

To validate our numerical model we proceed with the following approach. In the paper of [9], a lognormal probability function is built using the results of 324 free-falling cylinders measurements: 3 different lengths were combined with 3 different positions of the center of mass (COM)[7] and 7 different angles[8] for a total of 63 different test cases. Each of these set was then repeated to sum up the final 324 measurements. The measured landing distance - w.r.t. the dropping point - were then ordered by increasing order, and to each of them a *cumulative probability*

$$p = \frac{m}{N+1} \tag{3.2}$$

[7] due to symmetry reasons, one below, one above and one coincident with center of buoyancy

[8] 0°,15°,30°,45°,60°,75°,90°

was assigned, where N is the total number of simulations and m_i is the index number of the i-th simulation ($m_i = 3$ means that landing point of the i-th simulation was the third smallest one in order of magnitude). Each *cumulative probability* (equation 3.2) is then converted through the normal distribution into a value of standardised normal variable Z. In this way to each landing distance r is assigned to a standardised normal variable Z. For the experimental data a linear least-squares regression or regression on ordered statistics is used: this technique involves finding a probability and data scale that plots the cumulative distribution function (CDF) of a hypothesised distribution as a straight line (see Fig. 1.2). The corresponding linearity of the CDF for the sample data provides a measure of the goodness-of-fit of the hypothesised distribution. What we will do is to use the equation obtained for the experimental data

$$Z = 0.103e^{0.8147r} \tag{3.3}$$

as a benchmark for our numerical data. More precisely 10 different (see Tab.3.6[9]) numerical simulations will be made and their (Z_i, r_i) plot on log scale[10] will be compared with equation 3.3, in a plot similar to the one in Fig.1.2. For the validation test case the **Level 2** from Tab.3.5 has been

Simulation	**L/D**	**COM**	**Angle ϕ**
1	1.5	0	0°
2	1.125	+	0°
3	1.875	-	15°
4	1.5	+	15°
5	1.125	-	45°
6	1.875	+	45°
7	1.5	-	75°
8	1.125	0	75°
9	1.875	0	90°
10	1.5	0	90°

Table 3.6: Numerical Simulations

chosen: with limited amount of time and the intention to try and reproduce

[9]for COM: + is above, - is below and 0 coincide with the center of buoyancy

[10]y-axis $\mathbf{r}$, x-axis $\log_{\mathbf{e}}(\mathbf{Z})$

at least the main features of how to build a validation test case, that level of refinement is the only one who could have been run for ten simulations[11] in an affordable amount of time. As it is shown in next section, final result differs consistently with the experimental ones, being the mesh used still far from convergence.

3.4.2 Trajectory Extrapolation and landing points

```
import numpy as np
import io
import matplotlib.pyplot as plt
import os
import math
import re


with open('log.txt', 'r', encoding = 'utf-8') as f, open('com.dat','a') as f1
    for line in f:
        if 'Centre of mass:' in line:
            vect =re.split(r'[\s()]',line)
            f1.write(vect[8] + ' ' + vect[9] + ' ' + vect[10] + '\n')

Name = 'com.dat'
DATA = np.genfromtxt(Name)
x = DATA[:,0]
y = DATA[:,1]
z = DATA[:,2]
```

Figure 3.11: Python code used to obtain particle trajectory out of OpenFOAM output *log* file

To obtain the profile of particle trajectory, the strategy adopted is the one of *minimum effort - maximum result*: since by default the code is already outputting the *Centre of mass* value in terminal, the idea is simply to flash all the buffer into a log file, that is parsed using the Python code in Fig.3.11, in order to retrieve all the coordinates of the centre of mass of the object at each time step. Following the step described in previous section the plot of landing radius, against the corresponding cumulative probability distribution is shown in Fig.3.12. Once measured the radius hence, to each of the corresponding probability a standardized normal variable Z can be associated, following the normal probability distribution function. A further plot of landing points is proposed in Fig.3.13. The final step consists in plotting the obtained results (Z_i, r_i) and confronting them with the equation 3.3 to see that the amount of data generated numerically is not only very low in

[11] which is still a very low number if a statistics wants to be built

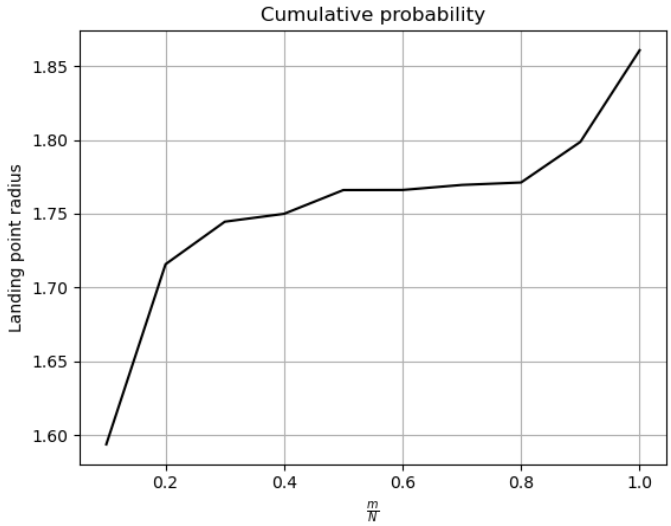


Figure 3.12: Landing radius cumulative probability function

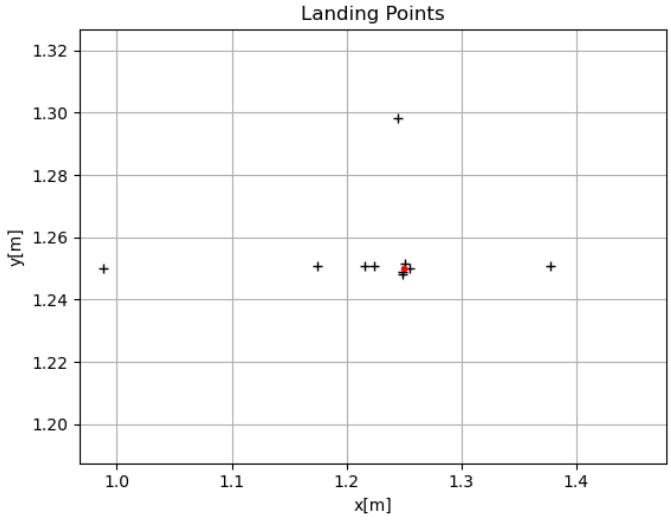


Figure 3.13: Landing Points on x-y plane: black crosses are landing points, red dot is dropping point

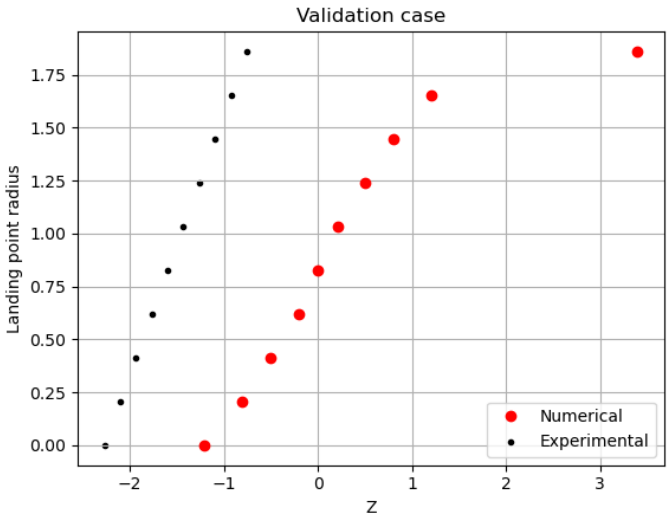


Figure 3.14: Validation of numerical data

numbers, but mostly not accurate enough to faithfully reproduce the results obtained experimentally (see Fig.3.14).

Chapter 4

Conclusion

In this project the physics of free falling cylinder crossing two different phases (air and water) has been studied. Few numerical methods suitable to reproduce the physics at stack were proposed, the final choice fell on the *overInterDyMFoam* algorithm, which foresees the use the *Overset Grid* algorithm, able to accurately capture the interaction between solid and fluid through multiple grids and interpolation procedures.The software used, and in which all the above mentioned algorithms are already present, is OpenFOAM-v2012 A further work has to be done to understand how to start from a floating cuboid tutorial test case, and arrive to a free falling cylinder: so not only handling degrees of freedom but also introduce the possibility to model different geometries. A sensitivity analysis has been carried out with the goal to retrieve the most accurate solution possible of a trajectory profile and landing point. Two main strategies are proposed, but none of them was actually able to beat the very high computational costs required to run a fully mesh-converged simulation. Some less accurate results were anyway used to sketch a possible validation procedure for the experimental work the whole project is based on. Further conclusions were pulled for what concerns the interchangeable role of two main RANS model for turbulence, i.e. the $k-\varepsilon$ and the $k-\omega$ models, as well as for the not irrelevant role that the external mesh plays on the final trajectory result.

Bibliography

[1] J. Davidson C. Windt B. Akram. "Performance assessment of the overset grid method for numerical wave tank experiments in the OpenFOAM enviroment". In: *International Conference on Ocean, Offshore and Arctic Engineering* (2018).

[2] *Floating Object Test-Case in OpenFAOM v2012*. URL: https://github.com/OpenFOAM/OpenFOAM-6/tree/master/tutorials/multiphase/interFoam/RAS/floatingObject.

[3] J. Chiva O. Lemkuhl I. Rodriguez. "Unsteady forces on a circular cylinder at critical Reynolds numbers". In: *Physics of fluids* (2014).

[4] *OpenFOAM boundary and initial conditions guide*. URL: https://www.openfoam.com/documentation/user-guide/standard-boundaryconditions.php.

[5] *overInterDyMFoam algorithm*. URL: https://www.openfoam.com/documentation/guides/latest/api/overInterDyMFoam_8C.html#details.

[6] *PIMPLE algorithm*. URL: https://www.simscale.com/forum/t/cfd-pimple-algorithm/81418.

[7] *Rotor Test-Case in OpenFAOM v2012*. URL: https://develop.openfoam.com/Development/openfoam/-/tree/master/tutorials/incompressible/overPimpleDyMFoam/twoSimpleRotors.

[8] *SIMPLE algorithm*. URL: https://openfoamwiki.net/index.php/SimpleFoam.

[9] S. Yasseri. "Experiment of free-falling cylinders in water". In: *The International Journal of the Society for Underwater* (2014).