Politecnico di Milano

Scuola di Ingegneria Industriale e dell'Informazione
Dipartimento di Meccanica

Corso di Laurea Magistrale in Ingegneria Meccanica

CFD modeling of cold-flow in the Darmstadt optical engine under steady-state and full-cycle conditions

Relatore:	Prof. Tommaso Lucchini
Correlatore:	Ing. Augusto Della Torre

Tesi di Laurea di:

Simona Belà Matr. 819354

Anno Accademico 2015-2016

Contents

List of Figures

List of Tables

Abstract

Details of the in-cylinder flow motions play a key role in determining the engine performance. Flow motions within the engine cylinder have a great influence on the fuel-air mixing, on the start and development of the combustion process, on the pollutant production and on heat transfer. The prediction of in-cylinder flow motion characteristics in function of the engine design parameters is thus of central importance. In the last decades the use of CFD (Computational Fluid Dynamics) has allowed to analyse the evolution of the in-cylinder flow motions and to perform parametric studies without the relative experimental costs.

In the present work two types of cold-flow simulations have been performed using OpenFOAM, an open-source CFD software, and Lib-ICE, a code based on OpenFOAM technology focused on internal combustion engine simulations developed by ICE Group of Politecnico di Milano. The first simulation analyses the mid-intake phase in steady-state condition, the second one instead is a full-cycle simulation. In both simulations a comparison between the two configurations of the TU-Darmstadt optical engine has been performed, in order to determine their capability to generate (and concerning the full-cycle simulation to maintain) a high Tumble motion. The achieved results have been validated with the available experimental data. The full-cycle simulation has been perform on one engine configuration only, then the results have been compared with those of a previous work carried out by the ICE Group on the other configuration.

Keywords: CFD, OpenFOAM, Lib-ICE, cold-flow, steady-state, full-cycle, Tumble, engine

Sommario

I dettagli relativi alle condizioni di moto del fluido nel cilindro giocano un ruolo fondamentale nella determinazione delle prestazioni del motore. I moti della carica controllano infatti il miscelamento del combustibile con l'ossidante, l'avvio e lo sviluppo del processo di combustione, la formazione di inquinanti e lo scambio termico. Risulta quindi essenziale poterne prevedere le caratteristiche in funzione dei parametri di progetto del motore. Negli ultimi decenni l'uso della CFD (Computational Fluid Dynamics) ha permesso di indagare l'evoluzione dei moti della carica all'interno del cilindro e condurre analisi parametriche senza i relativi costi sperimentali.

In questo lavoro sono stati effettuati due tipi di simulazioni cold-flow tramite OpenFOAM, un codice open-source per analisi CFD, e Lib-ICE, una libreria basata su OpenFOAM e creata dall'ICE Group del Politecnico di Milano specificamente per le simulazioni inerenti ai motori a combustione interna. La prima simulazione studia la fase di metà aspirazione in condizioni stazionarie, la seconda simulazione invece copre l'intero ciclo motore. In entrambe le simulazioni sono state confrontate le due configurazioni del motore ottico di TU-Darmstadt, al fine di determinare quale delle due sia in grado di generare (e nel caso della simulazione dell'intero ciclo motore sostenere) un moto vorticoso di Tumble più intenso. I risultati ottenuti sono validati tramite confronto con i dati sperimentali ove disponibili. La simulazione coprente l'intero ciclo motore è stata svolta su una sola delle due configurazioni, i risultati sono poi stati confrontati con i risultati di un lavoro precedente dell'ICE Group svolto sull'altra configurazione del motore.

Parole chiave: CFD, OpenFOAM, Lib-ICE, cold-flow, stazionario, ciclo completo, Tumble, motore

Introduction

Nowadays the internal combustion engine technology is of central importance in everyday life. Due to rising environmental trouble however, it is seen by the global community as one of the greater sources of pollutant emissions in the urban areas: the focus on pollutant production leads to a great deal of effort of the R&D field into the improvement of after-treatment techniques and combustion process itself, with the aim of a pollutant reduction (alongside with performances improvement) [4].

To achieve the awaited combustion process efficiency, great attention has to be paid to the charge motions into the cylinder. These organised gas motions have a great influence on the fuel-air mixing, on the start and development of the combustion process, on the pollutant production and on the thermal efficiency [3, 8].

For this reason, the design of the intake and exhaust ports together with the design of the combustion chamber itself and the piston crown, is studied specifically to enhance structured gas motions inside the cylinder [7]. Gas motion optimization studies by means of experimental test would lead to prohibitive costs, since a great number of prototypes would be required. In the last decades, a new powerful technique has been developed to analyse systems involving fluid flows: Computational Fluid Dynamics (CFD). This technique allows to perform parametric studies without the necessity of expensive experimental tests and provides a great level of results detail. These and much more advantages make CFD-analyses suitable for performance optimization.

In spark-ignition engines, both fuel-air mixing and combustion processes are driven by the complex flow features originating during the intake stroke and also residual gas distribution might affect combustion at partial load. For this reason it is necessary to model the gas exchange during the entire engine cycle, including the exhaust phase. The generation of high quality grids covering the full engine cycle is thus required and new fully automatic approaches are now available [16–18].

In the present work, a steady-state and a full-cycle CFD simulations are performed on the geometrical model of the TU-Darmstadt optical engine. It is a four-valve, pent-roof combustion chamber with flat piston, whose intake ducts are designed to generated a high Tumble motion. The TU-Darmstadt engine has quartz glass cylinder and piston allowing optical access of in-cylinder flow motions: experimental data of the flow fields are available.

Previous analyses were performed on the standard configuration of the optical engine. The current configuration is slightly modified: changes concern intake ducts orientation and valve dimensions. This work has two objectives. The first one is to verify the ability of OpenFOAM, an open-source CFD software, to predict the flow field, comparing the results with the experimental data. The second one is to analyse the capability of the current engine configuration to generate the Tumble motion, comparing the numerical results in the two engine configurations.

The first chapter is concerned with the fundamentals of Computational Fluid Dynamics (CFD): a brief introduction to this new powerful technique is followed by the fundamentals of fluid flows, turbulence models and discretization process.

In the second chapter a scheme of the structure of OpenFOAM is provided, together with the one of the Lib-ICE, a code based on OpenFOAM technology focused on internal combustion engines developed by ICE Group of Politecnico di Milano. Then the mesh management and the cold-flow simulation solver are explained.

The third chapter is concerned with the actual application of the full-cycle simulation to the engine under study. First the analysed engine is described in detail, then the automatic procedure for the generation of the full-cycle set of meshes and the following cold-flow simulation is explained in detail. Particular attention has been paid to the geometry requirements and the setting of the different steps of the case set-up.

In the fourth chapter the mesh generation is described in detail, from the choice of the grid type and resolution to the difficulties encountered to achieve a high quality grid. The achieved full-cycle set of meshes is thus shown.

The fifth chapter regards the gas-exchange simulation under steady-state condition, carried out in correspondence of the mid-intake phase. This type of simulation has been performed first on the standard configuration, where steady-state experimental data on the valve-plane were available for a validation [11]. Two types of sensitivity analysis have been carried out: turbulence model and numerical schemes. Then the same simulation has been performed on the current engine configuration, to allow a meaningful comparison between the capabilities of the two configurations of generating the Tumble motion.

Finally, in the sixth chapter the full-cycle cold-flow simulation is reported. The predicted flow field has been compared with experimental data available in correspondence of the mid-intake phase, verifying the validity of the employed methodology and the grid quality. Then the achieved results have been compared with results coming from a previous work on the standard configuration [16], in order to perform not only a numerical comparison but above all a motoring comparison between the two engine configurations and their capabilities to generate the Tumble motion.

Chapter 1

Computational Fluid Dynamics

1.1 Introduction to CFD

Computational Fluid Dynamics (CFD) is part of the CAE (Computer-Aided Engineering). CFD is the analysis of systems involving fluid flow, heat transfer and associated phenomena such as chemical reactions by means of computer-based simulation [27]. This technique is very powerful and spans a wide range of industrial and non-industrial application areas, such as: aerodynamics of aircraft and vehicles, flows and combustion in internal combustion engines (ICE), flows inside turbomachinery, wind loading on buildings and so on.

From the 1960s onwards the aerospace industry has integrated CFD techniques into the design, R&D and manufacture of aircraft and jet engines. Only more recently this approach has been applied to the design of ICE, gas turbines and furnaces. Since the 1990s, the availability of high-performance computing hardware and the introduction of user-friendly interfaces have led to the spread of CFD across the wider industrial community.

This huge diffusion of CFD in the industrial field can be explained considering the several advantages of this technique over experimental-based approach to fluid systems design:

- reduction of lead times and above all costs of new designs
- possibility to analyse complex and large systems where experiments are difficult, when not impossible, to perform
- ability to study systems under hazardous conditions at (and beyond) their normal performance limits
- great level of results' detail

- ability to cheaply perform parametric studies, making it suitable for performance optimization

CFD codes are based on the numerical solution of partial differential equations (PDE) describing the fluid behaviour (conservation of mass, momentum and energy). In addition to these equations, other conservation equations may be added to the code according to the analysed system requirements (as the species mass conservation).

As the system to be studied may be really complex, for the physical process involved and/or for the geometry, a mathematical modeling is required before a numerical solution strategy. This modeling step is the reason why CFD could never fully substitute experiments, which are necessary to validate the CFD code itself.

1.2 CFD applied to internal combustion engines

As already mentioned, the analysed system may be extremely complex: internal combustion engines (ICE) are one of the most complicated area to which CFD is applied nowadays.

There are many difficulties, starting from the geometry. The optimization of the performance of an ICE goes through the shape of its intake and exhaust port, the piston crown, the combustion chamber itself and many other systems according to the engine considered (spark-plug, spray-injector and so on). Not to mention the movement of the piston and its tuning with the valve motion. Therefore, the CAD description of the actual geometry must be as accurate as possible, and at the same time must fulfill the strict requirements of the CFD software (see chapter 3.2.1). The greater difficulty however is represented by the different interacting physical and chemical processes taking place inside an engine:

- non-stationary turbulent flows, pressure waves
- injection, atomization and evaporation of liquid fluid, mixing of gaseous fuel and fresh air
- ignition and combustion
- pollutants formation and handling

In the present work the attention is focused on the in-cylinder flow motion. Organised gas motions are shown in figure (1.1).

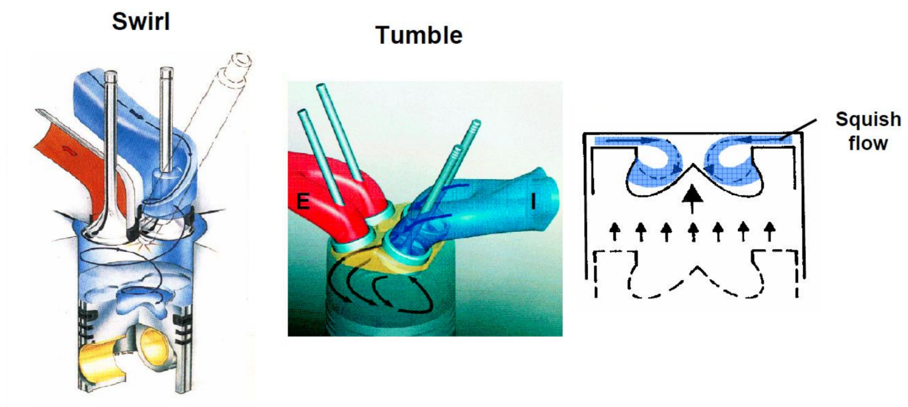


Figure 1.1: Organised gas motion: Swirl, Tumble, Squish.

These organised gas motions have a great influence on the fuel-air mixing, on the start and development of the combustion process, on the pollutant production and on the thermal efficiency. For this reason, their prediction is of central interest [3,13]. Among the organised gas motions shown in figure (1.1), the Tumble motion is the one analysed in this work.
Tumble is a structured rotational flow on a cylinder axial plane, generated during the intake stroke process, but maintained and increased by the following compression stroke. This motion was designed for the four-valve, pent-roof combustion chambers (for which swirl is not exploitable because of the symmetry of the cylinder head). The air flow, interacting with the cylinder walls and the piston head, undertakes a "tumble" which reverses its movement direction and organizes the flow in a structured vortex on an axial plane. Then the vortex size is decreased by the piston, as it moves towards top dead center (TDC): the angular momentum of the vortex is conserved and, since its radius is decreased, its angular velocity must increase. It is used in direct-injection engines to enhance fuel-air mixing and in port fuel injection SI engines to generate turbulence which increase the turbulent flame speed [8]. An explanation of Tumble motion is shown in figure (1.2).

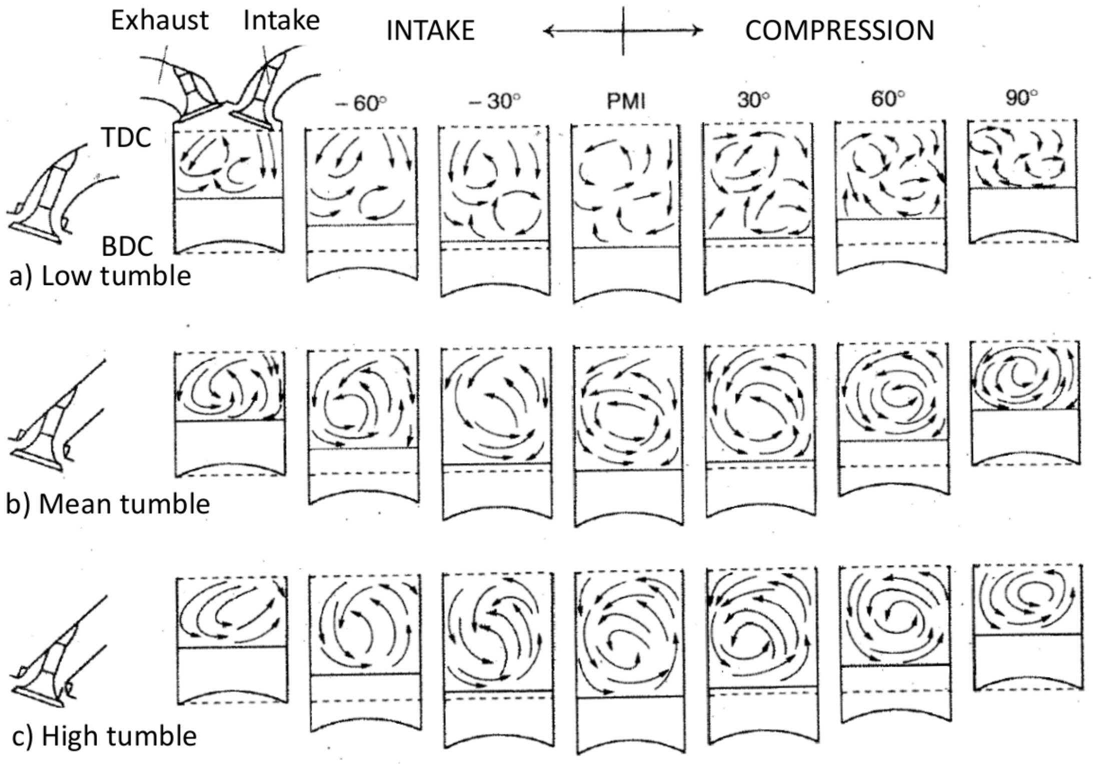


Figure 1.2: Tumble motion imposed by different intake ducts configurations.

The Tumble motion is measured, for each valve lift, under steady-state condition. The flow bench is shown in figure (1.3).

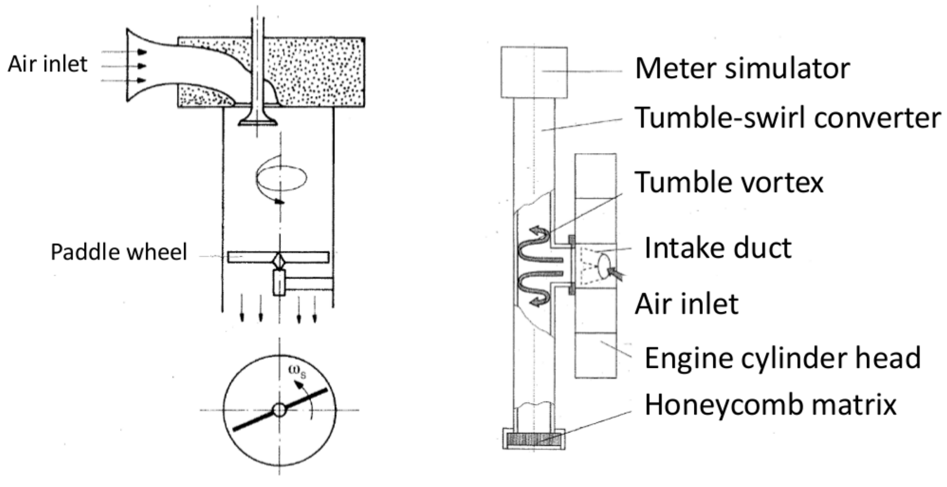


Figure 1.3: Flow bench for the Tumble measurement.

A kinematic measure of the Tumble motion, the Tumble ratio R_T, can thus be defined:

$$R_T = \frac{\omega_T D}{v_{is}} \tag{1.2.1}$$

where ω_T represents the paddle wheel rotation speed, D is the cylinder bore and v_{is} represents the velocity due to an ideal iso-entropic expansion under the pressure difference across the valve. The numerator represents the characteristic velocity of the Tumble motion, while the denominator is a velocity representing the axial flow motion.

Due to the complexity of the processes taking place inside an engine, simplified models of the underlying physics are required, from the turbulence model to the chemical one. Modeling is a tricky phase: when some aspects are neglected, results might be not accurate and reliable. Nevertheless, making some assumptions is essential to reduce the complexity to a manageable level whilst preserving the main aspects of the problem, leading to acceptable computational costs.

In ICE problems, models span from turbulence to ignition, from atomization to chemical reactions. In the following, governing equations and later on their discretization and numerical solution strategy are approached.

1.3 Governing equations

As previously mentioned, each CFD code is based on the conservation equations of fluid-dynamics representing the ground for any thermo-fluid-dynamical problem. These equations can completely determine the behaviour of the fluid system without the need of any additional dynamic law:

- conservation of mass: **continuity equation**
- conservation of momentum (Newton law): **momentum equation**
- conservation of energy (first principle of thermodynamics): **energy equation**

In 3D, this is a 5-equations system, since momentum is a vectorial quantity. Additional information are necessary as closure of the system: in order to determine the state of the fluid, equations of state are used. Under the assumption of thermodynamic equilibrium (acceptable even at large fluid velocity if strong shockwaves do not occur), these equations provide the link between the four thermodynamic variables (ρ, p, e, T).

This set of conservation equations, when applied to viscous flow is known as the **Navier-Stokes equations**, when applied to an inviscid one is known as the Euler equations.

The central concept of each conservation equation is flux: the variation of the total amount of a quantity inside a given domain is equal to the balance between the amount of the quantity entering and leaving the considered domain, plus the contribution from eventual sources generating that quantity.
From this definition, it is possible to write a conservation equation.
For a generic scalar quantity ϕ:

$$\frac{\partial}{\partial t}\int_{\Omega}\phi d\Omega = -\oint_{S}\overrightarrow{F}\cdot d\overrightarrow{S} + \int_{\Omega}Q_v d\Omega + \oint_{\Omega}\overrightarrow{Q_s}\cdot d\overrightarrow{S} \tag{1.3.1}$$

where:

- Ω is the controlled volume domain
- S is the surface defining the volume domain
- $\overrightarrow{F}$ is the flux
- Q_v and Q_s are volume and surface sources respectively

Eq. (1.3.1) is the integral conservation form for a generic scalar quantity.
Applying the Gauss' theorem to the surface integrals, the conservation law in integral form can be written as:

$$\frac{\partial}{\partial t}\int_{\Omega}\phi d\Omega + \int_{\Omega}\overrightarrow{\nabla}\cdot\overrightarrow{F}d\Omega = \int_{\Omega}Q_v d\Omega + \int_{\Omega}\overrightarrow{\nabla}\cdot\overrightarrow{Q_s}d\Omega \tag{1.3.2}$$

Since this conservation equation is valid for any arbitrary volume Ω, it must be valid in any point of the flow:

$$\frac{\partial\phi}{\partial t} + \overrightarrow{\nabla}\cdot\overrightarrow{F} = Q_v + \overrightarrow{\nabla}\cdot\overrightarrow{Q_s} \tag{1.3.3}$$

which is the differential conservation form for a generic scalar quantity.
Starting from Eq.(1.3.3), it is possible to observe that fluxes appear only under space derivative term, i.e. under the divergence operator.
Fluxes can be generated from two contributions:

- convective contribution: the amount of the conserved quantity ϕ carried away or transported by the fluid flow $\overrightarrow{F}_c = \phi\overrightarrow{U}$. The convective flux is proportional to the velocity of the flow and has directional properties. It appears as a first order partial derivative term and represents a non linear term as the velocity field depends on the transported variables.

- diffusive contribution: the amount of the conserved quantity ϕ carried away or transported by the presence of its gradient $\overrightarrow{F}_d = -k\rho\overrightarrow{\nabla}\phi$. The diffusive flux appears as a second order partial derivative term, i.e. under the Laplace operator.

Writing the two types of flux separately ($\overrightarrow{F} = \overrightarrow{F}_c + \overrightarrow{F}_d = \phi\overrightarrow{U} - k\rho\overrightarrow{\nabla}\phi$), the differential conservation form becomes:

$$\frac{\partial \phi}{\partial t} + \overrightarrow{\nabla} \cdot \phi\overrightarrow{U} = \overrightarrow{\nabla} \cdot \left(k\rho\overrightarrow{\nabla}\phi\right) + Q_v + \overrightarrow{\nabla} \cdot \overrightarrow{Q_s} \tag{1.3.4}$$

where the above mentioned divergence and Laplace operators appear. It is important to pay attention to these operators, since they require different numerical approach (see section 1.5.1).

If the conserved property is a vector $\overrightarrow{\phi} = (\phi_x, \phi_y, \phi_z)$, the flux becomes a tensor ($\bar{\bar{F}}$ notation), as well as the surface source term (stress tensor), whereas the volume source terms are a vector:

$$\frac{\partial}{\partial t}\int_\Omega \overrightarrow{\phi}\, d\Omega = -\oint_S \bar{\bar{F}} \cdot d\overrightarrow{S} + \int_\Omega \overrightarrow{Q_v} \cdot d\Omega + \oint_S \bar{\bar{Q}}_s \cdot d\overrightarrow{S} \tag{1.3.5}$$

applying the Gauss' theorem:

$$\frac{\partial}{\partial t}\int_\Omega \overrightarrow{\phi}\, d\Omega = -\int_\Omega \overrightarrow{\nabla} \cdot \bar{\bar{F}} d\Omega + \int_\Omega \overrightarrow{Q_v} \cdot d\Omega + \int_\Omega \overrightarrow{\nabla} \cdot \bar{\bar{Q}}_s d\Omega \tag{1.3.6}$$

Eq. (1.3.6) is the integral formulation of a vector conservation law.
Proceeding as for the scalar conservation case:

$$\frac{\partial}{\partial t}\overrightarrow{\phi} + \overrightarrow{\nabla} \cdot \left(\bar{\bar{F}} - \bar{\bar{Q}}_s\right) = \overrightarrow{Q_v} \tag{1.3.7}$$

Eq. (1.3.7) is the differential formulation of a vector conservation law.

From these general formulations, the conservation equations of fluid-dynamics can be easily derived [21, 27].

Continuity equation: this conservation is referred to a scalar property $\phi = \rho$

$$\frac{\partial}{\partial t}\int_\Omega \rho d\Omega + \oint_S \rho\overrightarrow{U} \cdot d\overrightarrow{S} = 0 \tag{1.3.8}$$

$$\frac{\partial \rho}{\partial t} + \overrightarrow{\nabla} \cdot \left(\rho\overrightarrow{U}\right) = 0 \tag{1.3.9}$$

Eq. (1.3.9) is the continuity equation written in conservative form (the space derivatives are grouped under the divergence term).

Momentum equation: this conservation is referred to a vectorial property $\phi = \rho\overrightarrow{U}$. It is common practice to highlight the contributions due to the surface forces (pressure and viscous forces, i.e. stresses) as separate terms in this conservation law and to include the effects of volume forces (gravity, applied forces) as source terms.

The former type is the expression of the fluid deformability and, since it depends on the position and orientation of the surface it acts on, it must be represented by a tensor.
Assuming that the fluid is Newtonian:

$$\bar{\bar{\sigma}} = -p\bar{\bar{I}} + \bar{\bar{\tau}} \tag{1.3.10}$$

where $-p\bar{\bar{I}}$ represents the isotropic pressure component (being $\bar{\bar{I}}$ the identity tensor) and $\bar{\bar{\tau}}$ is the viscous shear stress tensor, which represents the internal friction force of fluid layers against each other. Newton's constitutive law suggests the following linear model:

$$\bar{\bar{\tau}} = \mu \left[\left(\frac{\partial U_j}{\partial x_i} + \frac{\partial U_i}{\partial x_j} \right) - \frac{2}{3} \left(\overrightarrow{\nabla} \cdot \overrightarrow{U} \right) \delta_{ij} \right] \tag{1.3.11}$$

Inside the control volume the internal forces cancel out(therefore they are not represented by volume integrals).
Starting from these observations, the integral equation of momentum conservation equation will be:

$$\frac{\partial}{\partial t} \int_{\Omega} \rho \overrightarrow{U} d\Omega + \oint_S \left(\rho \overrightarrow{U} \otimes \overrightarrow{U} \right) \cdot d\overrightarrow{S} = \int_{\Omega} \rho \overrightarrow{f_e} d\Omega + \oint_S \bar{\bar{\sigma}} \cdot d\overrightarrow{S} \tag{1.3.12}$$

where $\otimes$ represents the tensor product. Applying the Gauss' theorem:

$$\frac{\partial}{\partial t} \int_{\Omega} \rho \overrightarrow{U} d\Omega + \int_{\Omega} \overrightarrow{\nabla} \cdot \left(\rho \overrightarrow{U} \otimes \overrightarrow{U} \right) d\Omega = \int_{\Omega} \rho \overrightarrow{f_e} d\Omega + \int_{\Omega} \overrightarrow{\nabla} \cdot \bar{\bar{\sigma}} d\Omega \tag{1.3.13}$$

Leading to the the differential form:

$$\frac{\partial \left(\rho \overrightarrow{U} \right)}{\partial t} + \overrightarrow{\nabla} \cdot \left(\rho \overrightarrow{U} \otimes \overrightarrow{U} + p\bar{\bar{I}} - \bar{\bar{\tau}} \right) = \rho \overrightarrow{f_e} \tag{1.3.14}$$

Notice that $\bar{\bar{\tau}}$ depends on the gradient of velocity (1.3.11), thus resulting in a diffusion flux (Laplace operator, here not written explicitly) for the momentum.

Energy equation: this conservation is referred to a scalar quantity $\phi = \rho E$, where $E = e + \frac{1}{2}\overrightarrow{U}^2$ is the total energy (sum of fluid internal energy and its kinetic energy) per unit mass.
The energy equation is derived from the first law of thermodynamics, stating that the rate of change of energy of the fluid is equal to the rate of heat addition to the fluid plus the rate of work done on the fluid. According to the previous perspective:

- the volume source terms are the work of the volume forces f_e and the heat sources:

$$Q_v = \rho \overrightarrow{f_e} \cdot \overrightarrow{U} + q_H \tag{1.3.15}$$

- the surface sources Q_s result from the work done on the fluid by the pressure and the internal shear stress acting on the control volume surface:

$$\overrightarrow{Q_s} = \bar{\bar{\sigma}} \cdot \overrightarrow{U} = -p\overrightarrow{U} + \bar{\bar{\tau}} \cdot \overrightarrow{U} \tag{1.3.16}$$

- flux can be easily separated in its type:
  - convective: $\overrightarrow{F}_c = \rho E \overrightarrow{U}$
  - diffusive: $\overrightarrow{F}_d = -k\overrightarrow{\nabla}T$, representing the thermal conductivity of the fluid

All these contributions lead to the energy conservation equation, which in integral form becomes:

$$\frac{\partial}{\partial t}\int_{\Omega} \rho E d\Omega + \oint_S \rho E \overrightarrow{U} \cdot d\overrightarrow{S} = \oint_S k\overrightarrow{\nabla}T d\overrightarrow{S} + \int_{\Omega}\left(\rho \overrightarrow{f_e} \cdot \overrightarrow{U} + q_H\right) d\Omega + \oint_S \left(\bar{\bar{\sigma}} \cdot \overrightarrow{U}\right) \cdot d\overrightarrow{S} \tag{1.3.17}$$

while in differential form is written as:

$$\frac{\partial \rho E}{\partial t} + \overrightarrow{\nabla} \cdot \left(\rho E \overrightarrow{U}\right) = \overrightarrow{\nabla} \cdot \left(k\overrightarrow{\nabla}T\right) + \overrightarrow{\nabla} \cdot \left(\bar{\bar{\sigma}} \cdot \overrightarrow{U}\right) + \rho \overrightarrow{f_e} \cdot \overrightarrow{U} + q_H \tag{1.3.18}$$

Expressing the stress tensor in its components:

$$\frac{\partial \rho E}{\partial t} + \overrightarrow{\nabla} \cdot \left(\rho E \overrightarrow{U} - k\overrightarrow{\nabla}T + p\bar{\bar{I}} \cdot \overrightarrow{U} - \bar{\bar{\tau}} \cdot \overrightarrow{U}\right) = \rho \overrightarrow{f_e} \cdot \overrightarrow{U} + q_H \tag{1.3.19}$$

For compressible flows it is useful to express the conservation equation (1.3.19) with respect to enthalpy ($H = h + \frac{1}{2}\overrightarrow{U}^2$, $h = e + \frac{p}{\rho}$) instead of total energy:

$$\frac{\partial \rho H}{\partial t} + \overrightarrow{\nabla} \cdot \left(\rho H \overrightarrow{U} - k\overrightarrow{\nabla}T - \bar{\bar{\tau}} \cdot \overrightarrow{U}\right) = \frac{\partial p}{\partial t} + \rho \overrightarrow{f_e} \cdot \overrightarrow{U} + q_H \tag{1.3.20}$$

In the present case of study, no further conservation equations are required. When the physical problem is more complex, e.g. when it includes the spray evolution and the combustion process, the code must solved also species mass conservation equation.

1.4 Turbulence models

"Nearly all macroscopic flows encountered in the natural world and in engineering practice are turbulent. [..] Turbulence involves fluctuations that are unpredictable in detail, and it has not been conquered by deterministic or statistical analysis. However, useful predictions about it are still possible and these may arise from physical intuition, dimensional arguments, direct numerical simulations, or empirical models and computational schemes." [15] The most common definition of turbulence is a flow regime characterized by chaotic property changes. In particular Reynolds [22] discovered that turbulent features begin to appear at particular values of a nondimensional number, called *Reynolds number*, defined as:

$$Re = \frac{UL}{\nu} \tag{1.4.1}$$

where U and L are typical velocity and length scale of the flow, while ν is the kinematic viscosity of the fluid.

One of the most widely accepted concept in turbulence theory is the *energy cascade*, concept firstly introduced by Richardson in 1922 [24]. The main idea of this view is that turbulence can be considered as a superposition of eddies of different sizes. An *eddy* eludes a precise definition, but it is conceived to be a turbulent motion, localized within a region of size l, that is at least moderately coherent over this region. The region occupied by a large eddy can also contain smaller eddies. Due to their high Reynolds numbers, the largest eddies have a big chance to become unstable and break up into smaller eddies. In this way turbulent kinetic energy is transferred to smaller eddies. The energy cascade continues until the Reynolds number is sufficiently small that the eddy motion is stable and molecular viscosity is effective in dissipating the kinetic energy. Figure (1.4) shows a visual description of the energy cascade concept.

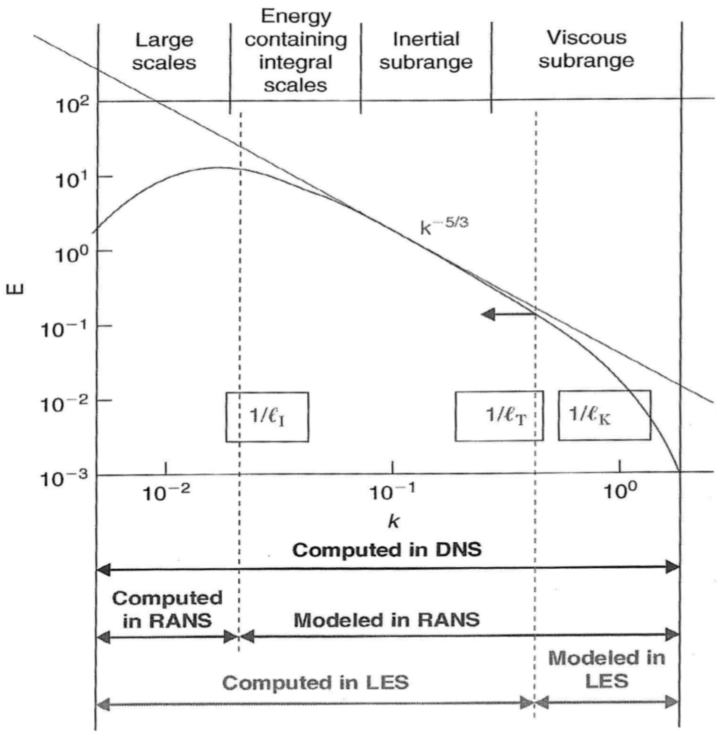


Figure 1.4: Energy cascade.

Specifically in ICEs, gas motion within the engine cylinder has a great influence on the fuel-air mixing and thus on the initial development of the combustion process. Nowadays the design of the intake and exhaust ducts, the piston crown and the valve timing are specifically studied in order to enhance organized large-scale motions (the largest eddies of the energy cascade concept), which are the effective source of turbulence motions inside the cylinder.

These large-scale gas motions are useful for both spark ignition and compression ignition engines. In the former type, swirl and/or tumble (structured large-scale motion induced during the intake stroke and increased during the compression one) are useful to enhance mixing between air, fuel and residual gases, to generate turbulence which increases the flame propagation speed and finally to achieve stratified mixture distribution. In the latter type, squish (structured large-scale motion induced during the compression and expansion strokes) is useful to promote fuel evaporation and mixing with air as well as to increase mixing between air and burned gases during the expansion stroke (allowing a better soot oxidation). These organized gas motions break down in smaller and smaller eddies, supplying them the kinetic energy required by turbulence.

For this reason, a correct modelling of large-scale vortex and of the consequent turbulence is fundamental in a CFD analysis of ICEs: numerical methods to capture turbulence effects are of central importance. Such methods can be grouped into three categories:

- Direct numerical simulation (**DNS**): all the turbulent scales, from the largest to the smallest are computed. In order to solve even the smallest turbulent velocity fluctuations, the spatial grid has to be extremely fine and the time steps small enough to resolve the fastest fluctuations. These requirements lead to excessive computational costs for the greatest part of industrial computations.
- Large eddy simulation (**LES**): this method involves space filtering of Navier-Stokes equations. Mean flow and large eddies are computed, while the small scale vortexes are modelled. The effects of unresolved eddies are included by means of a sub-grid scale model. Computational costs are still nearly prohibitive.
- Reynolds-averaged Navier-Stokes (**RANS**) equations: this method involves time averaging of Navier-Stokes equations: attention is focused on the mean flow and the turbulence effects on mean flow properties. Due to interactions between turbulent fluctuations, extra terms (which are new unknowns) appear in the time-averaged (or Reynolds-averaged) equations so that the effects of instantaneous fluctuations are not completely discarded. In particular, in the momentum equation appears a second order tensor -the Reynolds stress tensor-which according to Boussinesq is proportional to mean rates of deformation:

  $$\bar{\bar{r}} = -\rho\overline{\overrightarrow{u} \otimes \overrightarrow{u}} \tag{1.4.2}$$

  being $\overrightarrow{u}$ the velocity fluctuation. Using the suffix notation:

  $$r_{ij} = -\rho\overline{u_i u_j} = \mu_t \left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) - \frac{2}{3}\rho k \delta_{ij} \tag{1.4.3}$$

  where μ_t is the turbulent viscosity and $k = \frac{1}{2}\left(\overline{u^2} + \overline{v^2} + \overline{w^2}\right)$ is the turbulent kinetic energy. μ_t must be determined and this is the reason why turbulence models are required: in order to estimate the new unknowns generated by the averaging of Navier-Stokes equations, closure equations are necessary.

In figure (1.4) there is also a visual description of the scale application of these methods.
Concerning the specific application field of ICEs, RANS solvers are commonly used. Nevertheless, LES solvers find application in the analysis of the so called cyclic variability [26], which could not result using a time averaging procedure, while DNS can be applied in spray modelling [6].

In the present case, RANS solver is used. As previously described, it is necessary to develop turbulence models to predict the Reynolds stresses and close the system of mean flow equations. Different turbulence models are currently used in CFD codes (mixing length model, one equation models like Spalart-Allmaras, two equations models like k-ε and k-ω or even seven equations models). At present, the widely used and validated is the k-ε model, thanks to its robustness and reasonable accuracy for a wide range of turbulent flows. The unknown introduced by the time-averaging, the turbulent or eddy viscosity, can be expressed as follows:

$$\mu_t = \rho C_\mu \frac{k^2}{\varepsilon} \tag{1.4.4}$$

where C_μ is a dimensionless constant, k is the turbulent kinetic energy and ε the rate of dissipation of turbulent kinetic energy. The transport equations for k and ε are:

$$\frac{\partial \rho k}{\partial t} + \overrightarrow{\nabla} \cdot \left(\rho k \overrightarrow{U}\right) = \overrightarrow{\nabla} \cdot \left(\frac{\mu_t}{\sigma_k} \overrightarrow{\nabla} k\right) + \bar{\bar{r}} : \overrightarrow{\nabla} \overrightarrow{\overrightarrow{U}} - \rho\varepsilon \tag{1.4.5}$$

$$\frac{\partial \rho \varepsilon}{\partial t} + \overrightarrow{\nabla} \cdot \left(\rho \varepsilon \overrightarrow{U}\right) = \overrightarrow{\nabla} \cdot \left(\frac{\mu_t}{\sigma_\varepsilon} \overrightarrow{\nabla} \varepsilon\right) + C_{1\varepsilon} \frac{\varepsilon}{k} \bar{\bar{r}} : \overrightarrow{\nabla} \overrightarrow{\overrightarrow{U}} - C_{2\varepsilon} \rho \frac{\varepsilon^2}{k} \tag{1.4.6}$$

In both equations, the first term on the left side represents the rate of change of the variable (k or ε) and the second the transport of the variable by convection, while on the right side the terms represent in order the transport of the variable by diffusion, the rate of production and finally the rate of destruction of the variable. The ε-equation results problematic in the near-wall region since the term $\frac{\varepsilon^2}{k}$ is singular at the wall.

Another two-equations model, more accurate in the calculation of the near wall region, is the k-ω. In this model one equation is based on the specific dissipation rate $\omega = \frac{\varepsilon}{k}$, which can be considered a turbulent frequency. The transport equation for ω is written in similar form of the ε equation:

$$\frac{\partial \rho \omega}{\partial t} + \overrightarrow{\nabla} \cdot \left(\rho \omega \overrightarrow{U}\right) = \overrightarrow{\nabla} \cdot \left(\frac{\mu_t}{\rho \sigma_\omega} \overrightarrow{\nabla} \omega\right) + C_{1\omega} \frac{\omega}{k} \bar{\bar{r}} : \overrightarrow{\nabla} \overrightarrow{\overrightarrow{U}} - C_{2\omega} \rho \omega^2 \tag{1.4.7}$$

In the nineties, Menter proposed a modified version of the k-ω model, the k-ω SST, combination of the original two-equations models, with a blending factor making the model equivalent to the k-ω model close to the walls and to the k-ε model far from the walls.

Both the k-ε and the k-ω SST models are tested in the present work.

1.5 Discretization process

The aforementioned system of non-linear partial differential equations cannot be analytically solved. Discretization techniques allow to transform the partial differential equations into discrete algebraic equations numerically manageable. The major discretization methods are:

- Finite difference method (**FDM**)
- Finite element method (**FEM**)
- Finite volume method (**FVM**)

The most well-established CFD codes (both commercial like CFX/ANSYS, FLUENT and STAR-CD and open-source like OpenFOAM) uses the FVM technique for its great versatility and flexibility.
In figure (1.5) the various stages of the discretization process are illustrated [19].

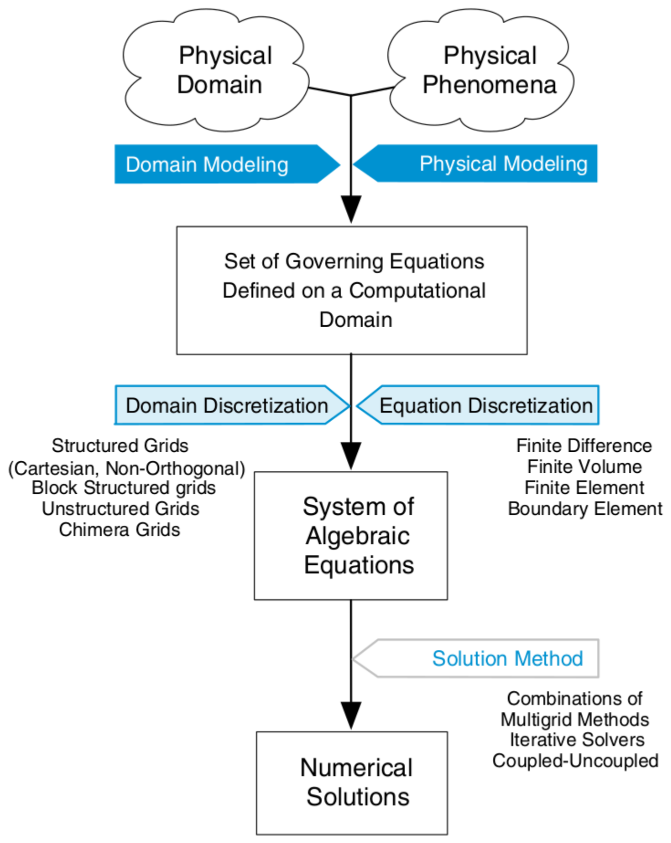


Figure 1.5: The discretization process.

Specifically, the FVM converts the partial differential equations representing conservation laws over differential volumes into discrete algebraic equations over finite volumes (or elements or cells).

The first step is the discretization of the computational domain into non-overlapping sub-domains: a grid or mesh of cells (control volumes) as the one shown in figure (1.6). The variables of interest are evaluated at the centroids of the control volumes, not at their boundary faces and this characteristic allows to implement boundary conditions without problems.

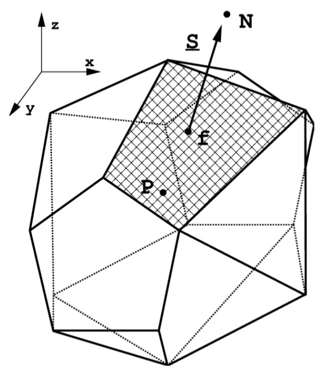


Figure 1.6: Control volume.

The centroid of the cell $\overrightarrow{x_P}$ is defined as:

$$\int_V (\overrightarrow{x} - \overrightarrow{x_P})\, dV = 0 \tag{1.5.1}$$

It is assumed that the property ϕ varies linearly within each cell:

$$\phi(\overrightarrow{x}) = \phi_P + (\overrightarrow{x} - \overrightarrow{x_P}) \cdot \left(\overrightarrow{\nabla}\phi\right)_P + O\left((\overrightarrow{x} - \overrightarrow{x_P})^2\right) \tag{1.5.2}$$

Similarly, the center of the face x_f is defined as:

$$\int_S (\overrightarrow{x} - \overrightarrow{x_f})\, dS = 0 \tag{1.5.3}$$

and linear variation of ϕ is assumed within each internal face:

$$\phi(\overrightarrow{x}) = \phi_f + (\overrightarrow{x} - \overrightarrow{x_f}) \cdot \left(\overrightarrow{\nabla}\phi\right)_f + O\left((\overrightarrow{x} - \overrightarrow{x_f})^2\right) \tag{1.5.4}$$

Then, the partial differential equations are discretized into algebraic equations by integrating them over each discrete element and finally iteratively solved.
Since the flux entering a given volume is identical to that leaving the adjacent volume across the common face, the FVM is intrinsically conservative, making it perfectly suitable for CFD problems.

1.5.1 Equation discretization

The equation discretization step is performed by integrating the differential equation over a control volume and then approximating the variation of the variable between mesh elements through imposed interpolation schemes. As the number of grid elements increases, the solution of the discretized equations approaches the exact solution of the corresponding differential ones, since as the control volumes get closer, changes in the variable between neighboring cells become small and thus the interpolation choice less influential.
More in detail, the problem is how to discretize each term of a generic transport equation. Starting from a generic transport equation in integral form with respect to a generic control volume (CV):

$$\int_{CV} \frac{\partial \rho\phi}{\partial t} dV + \int_{CV} \overrightarrow{\nabla} \cdot \left(\rho\phi\overrightarrow{U}\right) dV = \int_{CV} \overrightarrow{\nabla} \cdot \left(\Gamma\overrightarrow{\nabla}\phi\right) dV + \int_{CV} S_\phi dV \tag{1.5.5}$$

It is easy to highlight the various term: the rate of change and the convective terms on the left side, the diffusive (Γ is the diffusion coefficient) and the source term on the right side. Applying the Gauss theorem for the convective and diffusion terms:

$$\frac{\partial}{\partial t}\int_{CV} \rho\phi dV + \int_S \left(\rho\phi\overrightarrow{U}\right) \cdot \overrightarrow{n} dS = \int_S \left(\Gamma\overrightarrow{\nabla}\phi\right) \cdot \overrightarrow{n} dS + \int_{CV} S_\phi dV \tag{1.5.6}$$

with $\overrightarrow{n}$ outward normal vector.

Volume integral: the simplest approximation is to replace the volume integral by the mean value of the integrand (approximated by the cell center value) and the CV volume. This discretization is applied for example to the source term.

$$\begin{aligned}\int_{CV} \phi dV &= \int_{CV} \phi_P dV + \int_{CV} \left(\overrightarrow{x} - \overrightarrow{x_P}\right) dV \overrightarrow{\nabla}\phi + \int_{CV} O\left(\left(\overrightarrow{x} - \overrightarrow{x_P}\right)^2\right) dV \overrightarrow{\nabla}\overrightarrow{\nabla}\phi \\ &= \phi_P V_P + O|\left(\Delta x\right)|^2 \end{aligned} \tag{1.5.7}$$

The midpoint rule leads to an accuracy of the second order.

Surface integral: if convective and diffusive terms are present, the evaluation of surface integrals is necessary. The value of the integrand over the surface is required: the cell-face values are approximated in term of cell centroid values. A double approximation is necessary: the approximation of the integral using a finite number of face values (one, the face-center value, using the midpoint rule or three, the face center and the face vertexes, using the Simpson's rule, etc.) and the approximation

of the established face point(s) using interpolation schemes starting from the property value at the cell centroid.
As before, the midpoint rule leads to:

$$\int_S \phi dS = ... = \phi_f + O|\,(\Delta x)\,|^2 \tag{1.5.8}$$

However, the global accuracy depends not only on the midpoint rule, but also on the value of ϕ_f which must be obtain by interpolation.

Convection term: the convection term (or divergence term) is firstly transformed in a surface integrale using Gauss theorem:

$$\int_{CV} \overrightarrow{\nabla} \cdot \left(\rho\phi\overrightarrow{U}\right) dV = \int_S \left(\rho\phi\overrightarrow{U}\right) \cdot \overrightarrow{n} dS \tag{1.5.9}$$

Then, using the midpoint rule:

$$\begin{aligned}
\int_S \left(\rho\phi\overrightarrow{U}\right) \cdot \overrightarrow{n} dS &= \sum_f \overrightarrow{S} \cdot \left(\rho\phi\overrightarrow{U}\right)_f \\
&= \sum_f \overrightarrow{S} \cdot \left(\rho\overrightarrow{U}\right)_f \phi_f \\
&= \sum_f \left(S\rho U_n\right)_f \phi_f \\
&= \sum_f F\phi_f
\end{aligned} \tag{1.5.10}$$

The most used interpolation schemes are the **upwind** scheme, which approximates ϕ_f with the value of the neighboring cells on the basis of the flow direction (first order accuracy) or the **linear or central difference** interpolation, where ϕ_f is interpolated between the two nearest cell centroids (second order accuracy). The first scheme is less accurate but conditionally stable, while the second is more accurate but unstable: in case of sharp gradient stability is preferred to accuracy.
Other methods like QUICK or polynomial approximations have higher accuracy.

Gradient term: the gradient term, after the application of the Gauss theorem, needs a linear (or higher) interpolation, otherwise it cancels out.

Diffusive term: the diffusion term (or laplacian term) is first transformed in a surface integrale using Gauss theorem:

$$\int_{CV} \overrightarrow{\nabla} \cdot \left(\Gamma\overrightarrow{\nabla}\phi\right) dV = \int_S \left(\Gamma\overrightarrow{\nabla}\phi\right) \cdot \overrightarrow{n} dS \tag{1.5.11}$$

Then, as for the convection term:

$$\begin{aligned}\int_S \left(\Gamma\overrightarrow{\nabla}\phi\right)\cdot\overrightarrow{n}\,dS &= \sum_f \overrightarrow{S}\cdot\left(\Gamma_\phi\overrightarrow{\nabla}\phi\right)_f \\ &= \sum_f \left(\Gamma_\phi\right)_f\left(\overrightarrow{S}\cdot\overrightarrow{\nabla}\phi\right)_f \end{aligned} \tag{1.5.12}$$

where the term $\overrightarrow{S}\cdot\overrightarrow{\nabla}\phi$ is evaluated as:

$$\left(\overrightarrow{S}\cdot\overrightarrow{\nabla}\phi\right)_f = |\overrightarrow{S}|\frac{\phi_P-\phi_N}{|\overrightarrow{d}|} + \overrightarrow{k}\cdot\left(\overrightarrow{\nabla}\phi\right)_f \tag{1.5.13}$$

where ϕ_P and ϕ_N are the values of the property at the adiacent cell centroids and $\overrightarrow{k}$ is the correction that must be applied to evaluate the gradient in case of a non orthogonal mesh (see figure (2.7)).

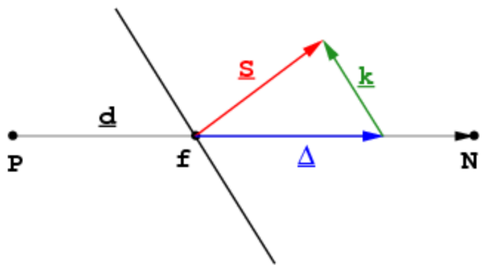


Figure 1.7: Gradient evaluation in case of non-orthogonality.

Time discretization: in case of time-varying problems an integration over time is also needed. Together with a domain discretization, a time discretizazion is required: the time domain is divided into time steps Δt. The time discretization methods can be divided into explicit and implicit. The former type is always stable (thus the time step can be larger), but computationally more expensive, while the latter type is only conditionally stable: in order to have stability the time step cannot be larger than a maximum value, which depends on the Courant number $Co = \frac{u}{\Delta x/\Delta t}$, where u is the magnitude of the velocity and Δx the grid dimension. To ensure stability, the condition $Co < 1$ must be respected. Among the implicit methods, the most used are **Euler** (accuracy of the first order) and **Crank-Nicolson** (second order).

Chapter 2

OpenFOAM and Lib-ICE

OpenFOAM (Open Source Field Operation and Manipulation) is first and foremost a C++ library, used primarily to create executables, known as *applications*. The applications fall into two categories: *solvers*, that are each designed to solve a specific problem in continuum mechanichs; and *utilities*, that are designed to perform tasks that involve data manipulation [20]. The great potentiality of this tool is not restricted to its free-access: new solvers and utilities can be created by its users and compiled in addition to standard ones.

OpenFOAM is supplied with pre- and post-processing environments. The interface to pre- and post-processing are themselves OpenFOAM utilities, thereby ensuring consistent data handling across all environments. The overall structure of OpenFOAM is shown in figure (2.1), while in figure (2.2) the structure of applications directory is explained.

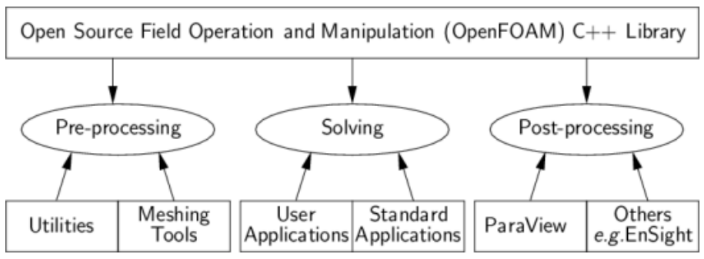


Figure 2.1: Overview of OpenFOAM structure.

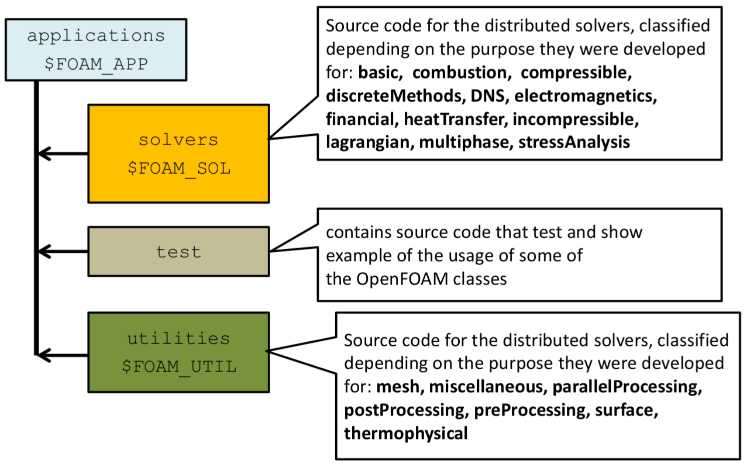

applications
$FOAM_APP
solvers
$FOAM_SOL
test
utilities
$FOAM_UTIL
Source code for the distributed solvers, classified depending on the purpose they were developed for: **basic, combustion, compressible, discreteMethods, DNS, electromagnetics, financial, heatTransfer, incompressible, lagrangian, multiphase, stressAnalysis**
contains source code that test and show example of the usage of some of the OpenFOAM classes
Source code for the distributed solvers, classified depending on the purpose they were developed for: **mesh, miscellaneous, parallelProcessing, postProcessing, preProcessing, surface, thermophysical**

Figure 2.2: Applications directory.

As previously introduced, the strength of OpenFOAM is multiple: its free-access allow and enhance cooperation among the growing community of users (academic but also commercial), making the spreading of ideas and new models easier; its open structure make it possible to implement personal libraries.

Thanks to this possibility, the ICE Group of Politecnico di Milano has developed the Lib-ICE: a code based on OpenFOAM technology focused on internal combustion engine simulations (figure 2.3).

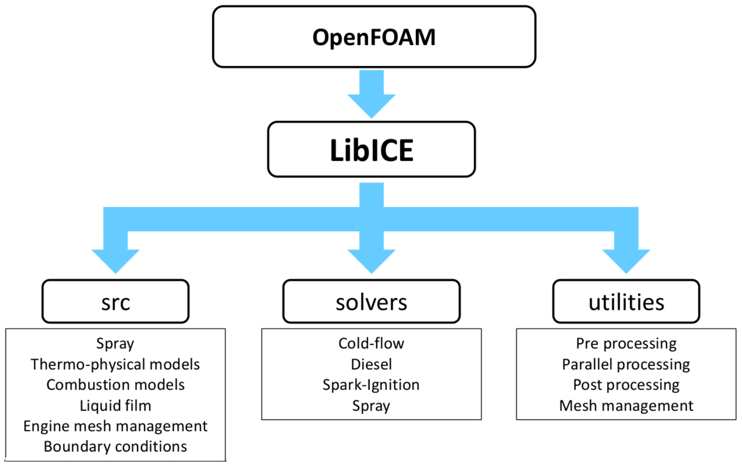
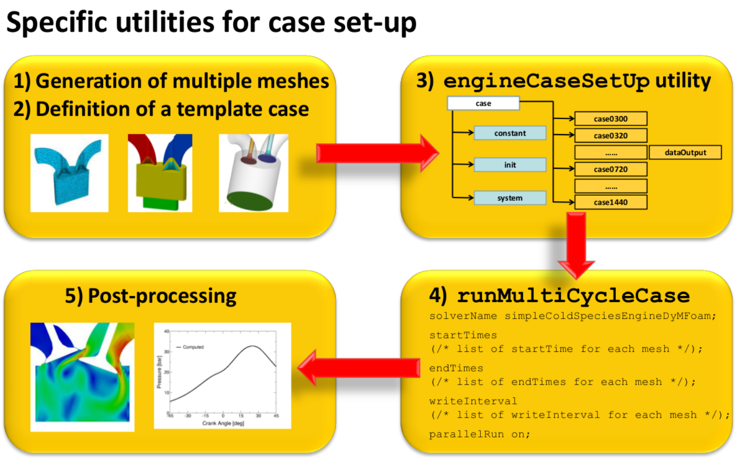


Figure 2.3: Lib-ICE structure.

Lib-ICE can handle automatic mesh generation for full-cycle simulations, fuel-air mixing modelling in GDI (gasoline direct injection) engines, diesel combustion, lagrangian spray modelling, after treatment modelling and much more.

In particular, specific utilities for in-cylinder full-cycle simulations are present, making it possible to easily run in parallel an entire engine cycle (figure (2.4)).



```
solverName simpleColdSpeciesEngineDyMFoam;
startTimes
(/* list of startTime for each mesh */);
endTimes
(/* list of endTimes for each mesh */);
writeInterval
(/* list of writeInterval for each mesh */);
parallelRun on;
```

Figure 2.4: Specific utilities for full-cycle case set-up.

2.1 Mesh management

In a full-cycle simulation, a unique mesh cannot cover the entire cycle of 720 crank-angle degrees: to cover the whole cycle, multiple meshes are required (figure 2.5).

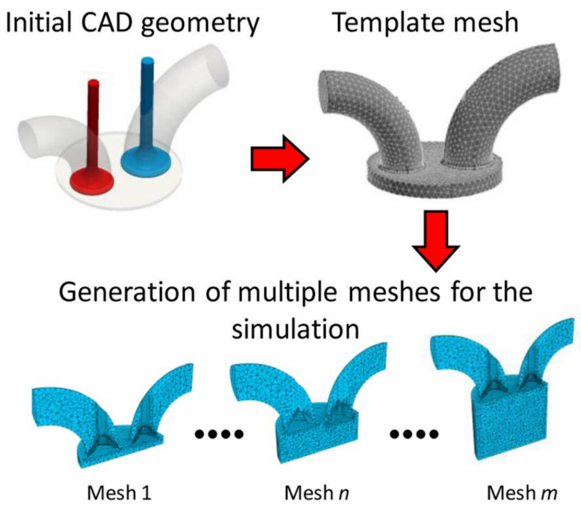


Figure 2.5: Multiple meshes are required to cover the entire engine cycle.

ICEs are particularly complex in their motion: piston and valves move simultaneously along different axes with different motion equations. Generally the computational grid is divided in different regions according to the geometrical domain and the motion strategy.

Boundary motion can be considered as given: in ICEs simulations, it is prescribed by piston and valve motion. The critical step is represented by the internal points motion: it must accommodate the computational domain changes (boundary motion) while preserving the validity (both topological and geometrical) and quality of the mesh. A grid is thus moved until quality requirements are satisfied, then a new grid must be generated. This procedure is repeated until the entire engine cycle is covered. The number of required meshes to cover the full cycle generally increases with the complexity of the geometrical domain.
There are two type of mesh motion strategy:

- **Automatic mesh motion**: it is based on pure deformation. The topology of the mesh does not change, connectivity remains the same. The internal points motion is determined by a vertex-based motion solver. This strategy is very powerful, fully automatic and it is not subject to limitations on the mesh complexity.
- **Topological changes**: it is based on the change of the number or connectivity of points, faces or cells in the mesh during the simulation. Dynamic mesh layering, sliding interface and attach/detach boundary belong to this motion strategy. Most of the mesh points remains unchanged, the topology is changed, e.g., by the addition/removal of cell layers (dynamic mesh layering).

2.1.1 Polyhedral vertex-based motion solver

In a full-cycle ICEs simulation, the automatic mesh motion is the most convenient choice. Practically it is performed using the polyhedral vertex-based motion solver. The internal points displacement is calculated by solving a motion equation having as boundary condition the prescribed boundary motion. Laplace equation is chosen as mesh motion equation and it is solved for the point velocity field $\overrightarrow{U_p}$ and the diffusivity γ, which can be uniform or variable:

$$\overrightarrow{\nabla} \cdot \left(\gamma \overrightarrow{\nabla} \overrightarrow{U_p}\right) = 0 \tag{2.1.1}$$

The new position of each internal point is then calculated as:

$$\overrightarrow{x}_{new} = \overrightarrow{x}_{old} + \overrightarrow{U_p} \Delta t \tag{2.1.2}$$

being $\overrightarrow{x}_{new}$ and $\overrightarrow{x}_{old}$ the points position after and before the mesh motion respectively, and Δt the time step.

As already said, mesh validity and quality must be preserved during the internal points motion. In order to avoid degenerated control volumes, the Laplace operator is discretized on a finite element decomposition of the polyhedral mesh: each polyhedral cell is split into tetrahedra by splitting its faces into triangles and introducing a point in the cell centroid (figure 2.6). Consistency in tetrahedral connectivity is obtained by using identical face decomposition for both cells sharing an internal face [14].

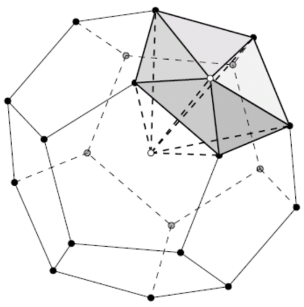


Figure 2.6: Decomposition of a polyhedral cell into tetrahedra.

The diffusivity field γ controls the mesh quality during the points motion. It is possible to set a uniform diffusivity or a variable one. Uniform diffusivity generally maintains mesh validity but leads to a poor mesh quality. The way to diminish local distortion is by increasing local diffusivity in the motion operator. Among the diffusion laws, in CFD codes are available the following formulations:

- **inverse point distance diffusivity**: a number of boundary patches are selected by the user and the diffusion field is a function of cell centroid distance d to the nearest selected bounudary:

$$\gamma = \frac{1}{d^\alpha} \tag{2.1.3}$$

- **inverse volume diffusivity**: the diffusion field is a function of the cell volume V:

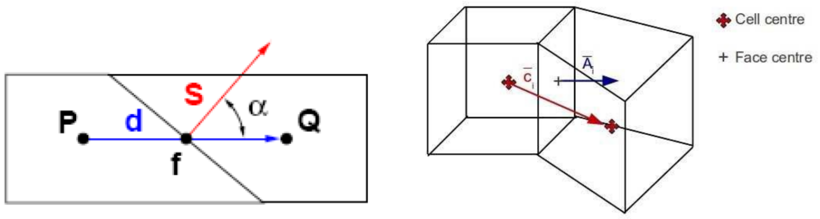
$$\gamma = \frac{1}{V^\alpha} \tag{2.1.4}$$

In both formulations, $\alpha > 0$.
In the present case of study, both inverse point distance and inverse volume diffusivity have been applied for each mesh motion, finally choosing the method leading to larger mesh validity.

Mesh quality indexes

It was highlighted that mesh quality is taken into account as crucial requirement during mesh motion. A good mesh quality indeed, together with a fine grid size, influences the computed solution: numerical methods accuracy is particularly affected by two quality indexes: *skewness* and *non-orthogonality*.

Non-orthogonality: it is an index of the alignment between the vectorial distance between the centroids of two neighbouring cells and the vector normal to the face. It is defined as the angle between these two vectors, and it should not overcome 70° (severe non-orthogonality). Non-orthogonality higher than 90° is not acceptable as it leads to degenerated cells. A representation of non-orthogonal mesh is shown in figure (2.7). Non-orthogonality affects the discretization accuracy of the diffusion



(a) (b)

Figure 2.7: Examples of non-orthogonal mesh in 2D (a) and 3D (b).

term in transport equations.

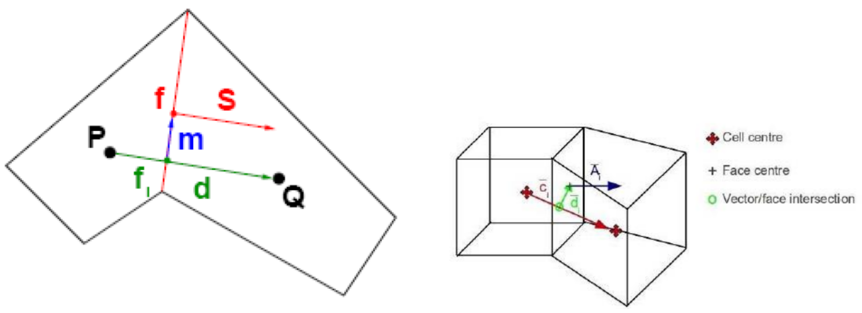
Skewness: it is an index of the distortion of the grid. It is calculated as the distance between the face center and the face intersection point of the vector connecting the centroids of the neighbouring cells, normalized by the distance between the centroids of the neighbouring cells. A representation of skew mesh is shown in figure (2.8).

$$skewness = \frac{|\overrightarrow{d}_i|}{|\overrightarrow{C}_i|} \tag{2.1.5}$$

Recommended maximum values are 20 for boundary skewness and 4 (but admissible till 10) for internal skewness.

Interpolation

Once the set of meshes has been generated, it is possible to run the simulation itself. The simulation is performed on the the first mesh up to its validity, then the computed field conditions (pressure, temperature, velocity, etc.) must be interpolated



(a) (b)

Figure 2.8: Examples of skew mesh in 2D (a) and 3D (b).

(mapped) on the next mesh, where the simulation continues. This procedure is schematically shown in figure (2.9). The starting mesh is the so called *source* mesh, while the next one is the *target* mesh.

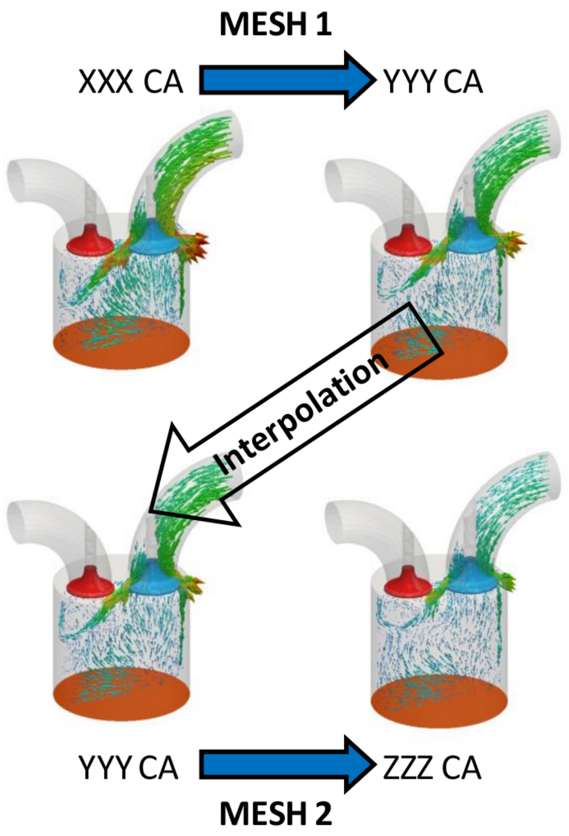


Figure 2.9: Strategy for full-cycle ICEs simulations: case to case interpolation.

The chosen interpolation method is a second-order inverse distance weighting: for each cell centroid on the target mesh, the closest cell centroid in the source mesh is identified (figure (2.10)).

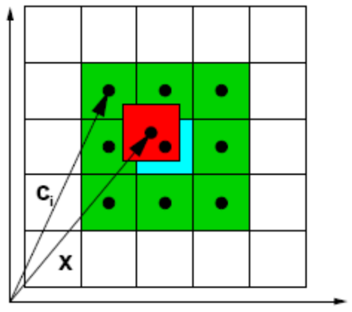


Figure 2.10: Identification of the closest (cyan) and neighbouring (green) cells in the source mesh for a cell in the target mesh (red).

As shown in figure (2.10), also the neighbouring cells must be taken into account: the field value for each cell centroid of the target mesh is computed as a function of the field values on the corresponding cell centroid and its neighbouring cells in the source mesh:

$$\phi(x) = \frac{1}{\sum \alpha_i} \sum \alpha_i \phi_i \qquad \alpha_i = \frac{1}{|x - c_i|} \tag{2.1.6}$$

where x represents the cell centroid on the target mesh, ϕ_i the value of the field on the source mesh and α_i (the weights) are evaluated using the distance between x and the cell centroids of the neighbouring cells in the source mesh c_i. If the distance between target and source cell centroids is lower than a defined tolerance, no interpolation is performed: the field value in the target cell is set equal to the one in the corresponding cell centroid in the source mesh.

2.2 Cold-flow simulation solver

In the present work, a full-cycle cold-flow simulation is performed. OpenFOAM provides different types of transient solvers for turbulent compressible flows (*rhoCentralFoam*, *rhoPimpleFoam*,*sonicFoam*), some of them include also optional mesh motion and mesh topology changes, like *rhoPimpleDyMFoam*. Lib-ICE however provides a specific utility to run full-cycle gas exchange simulations (**runMultiCycleCase**) perfectly coupled with the utility used to generate automatically the whole set of meshes (**engineDynamicSetUp**), as will be explained in the next chapter. Together with specific utilities, specific solvers are implemented, divided into the categories: *coldFlow*, *combustion*, *compressible*, *Diesel*, *multiphase* and *sparkIgnition*.

Among the coldFlow solvers, the **simpleColdSpeciesEngineDyMFoam** solver has been applied for this work.

The solver employs a PIMPLE algorithm, a coupling of SIMPLE and PISO algorithms. Actually it is nothing but a PISO with two addiction: outer correction loops (i.e. multiple cycling over the same time step using the last iteration final value as initial guess for the next iteration) and under-relaxation of the variables between consequent outer iterations (SIMPLE). The final result is a higher stability than the one achievable with a PISO algorithm, even with larger time step. In full-cycle simulations the possibility to use larger time step is crucial, as the simulation lasts 720 crank-angle degrees (CA). In figure (2.11) an outline of PIMPLE algorithm is shown.

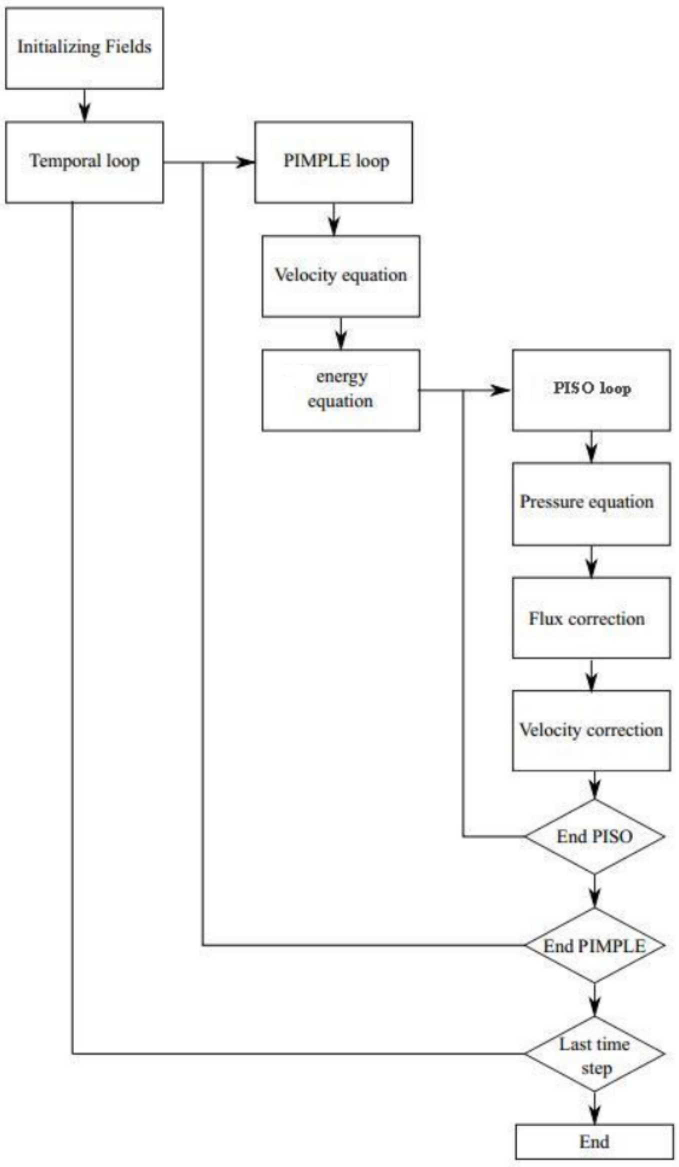


Figure 2.11: PIMPLE algorithm.

simpleColdSpeciesEngineDyMFoam is a solver for compressible flows: before the pimple-loop, density equation is solved, inside the pimple loop also the energy equation (in the present case in its enthalpy formulation) is solved and compressibility effects are taken into account in the pressure and velocity equations.

In addition, it includes the mesh motion: if the mesh is changed it applies proper corrections and if the mesh is moving it continuously update the mesh flux.

Finally it automatically writes outputs useful for post-processing analyses.

Chapter 3

Case of study and implementation

In this chapter, the implementation of the above explained simulation strategy on the case of study is deeply described. After a focus on the case of study - **TU-Darmstadt engine** -, settings of the mesh generation by means of the utility **engineDynamicSetUp** are reported, together with the problems occurred during the meshing step. The last part of this chapter concerns the settings of the cold-flow simulation, performed by means of the utility **runMultiCycleCase**.

3.1 TU-Darmstadt optical engine

The simulation is performed on a geometrical model of the TU-Darmstadt optical engine: a single-cylinder spark-ignition (SI) engine fully optically accessible. It is a square, four-valve, pent-roof engine with the following geometric parameters:

- conrod length: 148 [mm]
- bore: 86 [mm]
- stroke: 86 [mm]
- clearance: 2,6 [mm]
- revolutions per minute: 800 [rpm]

Figure (3.1) shows a CAD description with the experimental points of interest (provided by TU-Darmstadt).

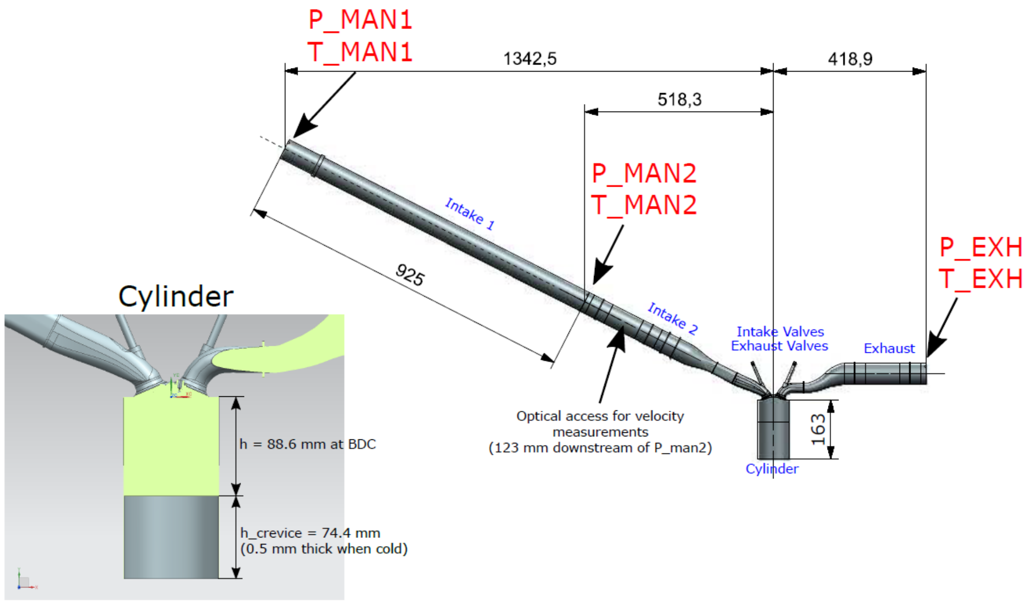


Figure 3.1: Engine CAD description.

As shown in figure (3.1), the CAD model includes also a crevice height: combustion chamber crevices in spark-ignition engines are identified as the largest contributors to the engine-out hydrocarbon emissions. Their modelling has thus great influence in the prediction of thermal efficiency and pollutants production. In the present work crevices have been included in the mesh generation in order to achieve a grid suitable for both the cold-flow and combustion process simulations. In the cold-flow simulation reported in this work, the crevices modelling has allowed to assess their effect on the engine compression ratio.

The optical access is provided by a 55mm height quartz-glass liner (20 mm thickness), an 8 mm window extension into the pent-roof, and a Bowditch-piston with flat quartz-glass piston-crown window (75 mm diameter). Additionally, optical access is granted into the straight piping of the intake manifold by a DN50 fused-silica cylinder [11]. The TU-Darmstadt optical engine is shown in figure (3.2).

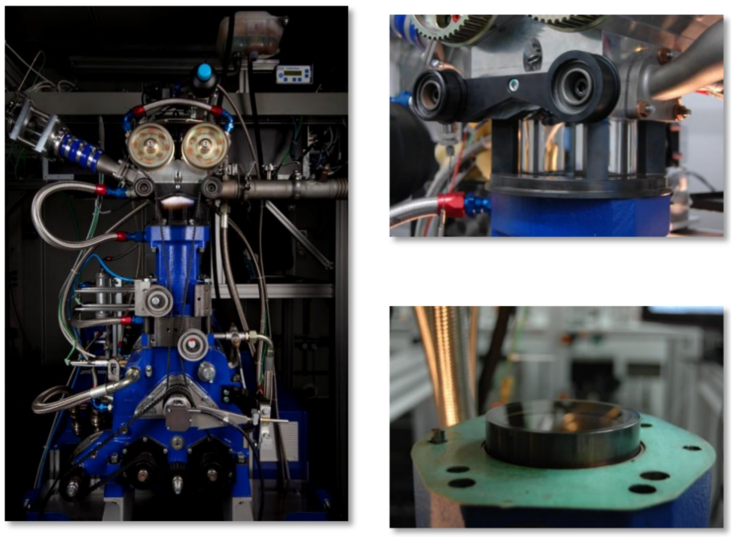


Figure 3.2: TU-Darmstadt optical engine.

At TU Darmstadt particle image velocimetry (PIV) measurements were performed at crank-angle degree (CA) 270 bTDC (before top-dead-center), corresponding to a mid-intake phase (exhaust valves closed, intake valve lift around 9 mm). Further experimental measurements performed at the same CA in steady-state conditions are available for the standard configuration (different intake ducts orientation, slightly bigger intake valves diameter). In this work a steady-state simulation has been performed on both the standard and the current engine configuration in order to achieve a meaningful experimental validation together with a performance comparison between the engine configurations.

Then a full-cycle simulation has been performed and the results has been compared to those of previous simulations referring to the standard engine configuration in correspondence of 450 (450 = 270 bTDC), 540 and 630 CA (that is mid-intake, intake in correspondence of BDC and mid-compression).

Both simulations have been performed on the same grid, whose generation procedure is deeply described in the following section.

3.2 engineDynamicSetUp

Generating high quality meshes is essential for CFD computations, the meshing step is thus of central importance.

The full-cycle set of meshes has been generated with a fully automatic utility implemented in the Lib-ICE library, **engineDynamicSetUp**. It is an application

for the dynamic creation and management of a complete set of meshes able to cover the entire engine cycle. The tool automatically creates the grids, prepares the case folders and manages the dictionaries for the full cycle simulation [9].
engineDynamicSetUp works in loop:

- generation of the stl file at the desired crank-angle degree (CA): **createEngineSTL** utility
- generation of the mesh: *snappyHexMesh* utility
- generation of the case folder and set-up of the dictionaries
- movement of the mesh till satisfies quality parameters: **moveEngineDynamicMesh** utility

In order to work properly, this utility requires two initial step:

- pre-processing of the input geometry: the CAD geometry in stl format must by divided in proper multiple surfaces (patches), necessary for the definition of the domain motion (figure (3.3)).

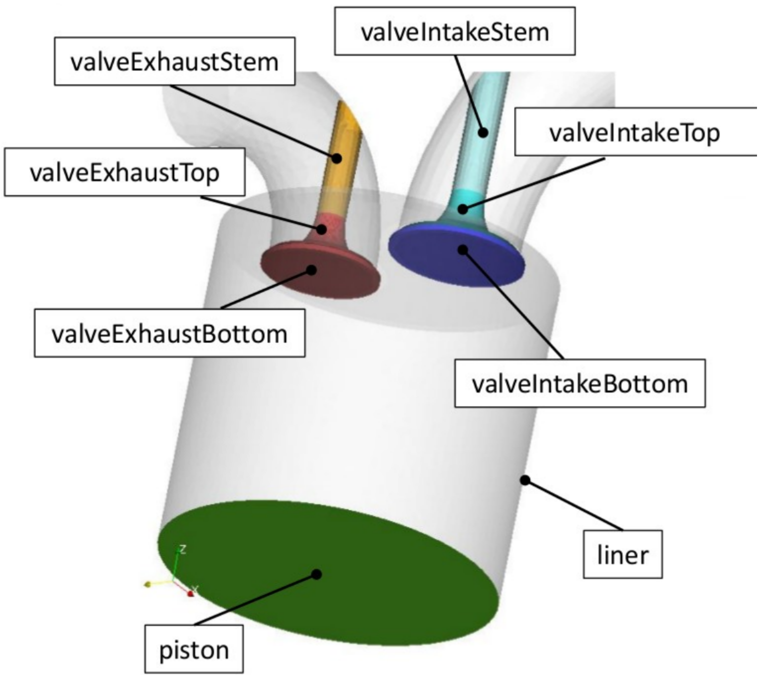


Figure 3.3: Division of the surface in patches, necessary for the definition of the domain motion.

To achieve a correct geometry motion, the input stl geometry must be positioned at the top-dead center (TDC) with all the valves at the minimum lift below which are considered closed (minLift).

- set-up of all the dictionaries required in the different case folders, from those concerning the mesh generation, to those for the mesh motion till the ones required for the following full-cycle simulation.

engineDynamicSetup entrusts *snappyHexMesh*, a mesh generation utility supplied with OpenFOAM, with the single-mesh generation. This utility generates 3-dimensional meshes containing hexahedra (hex) and split-hexahedra (split-hex) automatically from triangulated surface geometries, or tri-surfaces, in Stereolithography (STL) or Wavefront Object (OBJ) format. The mesh approximately conforms to the surface by iteratively refining a starting mesh and morphing the resulting split-hex mesh to the surface. The specification of mesh refinement level is very flexible and the surface handling is robust with a pre-specified final mesh quality [20].
snappyHexMesh requires high quality input geometry to allow a correct meshing.

3.2.1 Geometry

TU Darmstadt provided us CAD file generated using NX9, in prt, stl, stp formats. This geometry file had many features avoiding its use as tri-surfaces in the meshing tool and overall as computational domain for a CFD simulation. In figure (3.4) a representation of the original geometry.

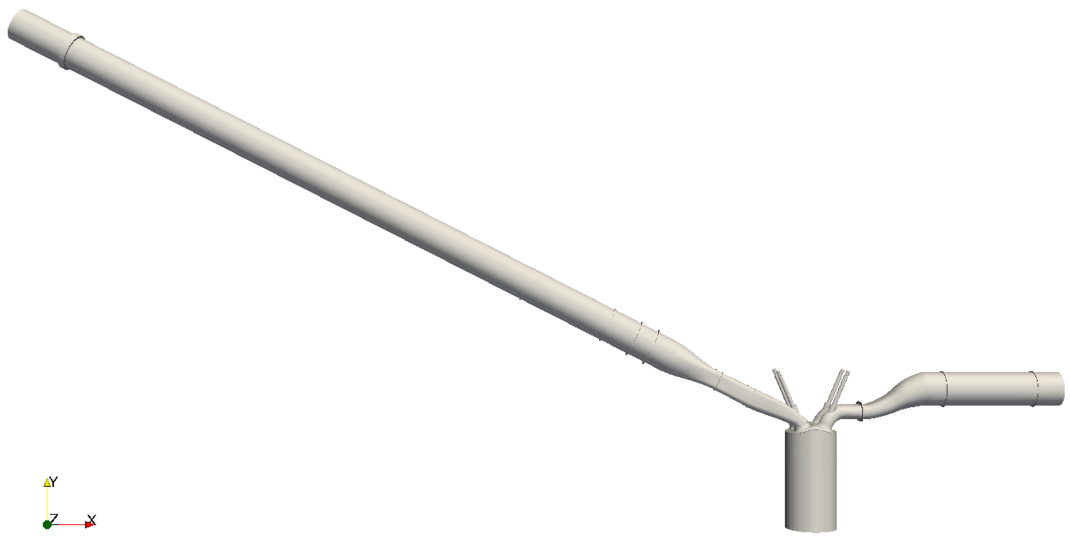


Figure 3.4: TU Darmstadt engine, STL file provided.

Thanks to figures (3.5) and (3.6), representing slices of the original geometry at difference planes, it is possible to notice the existing problems.

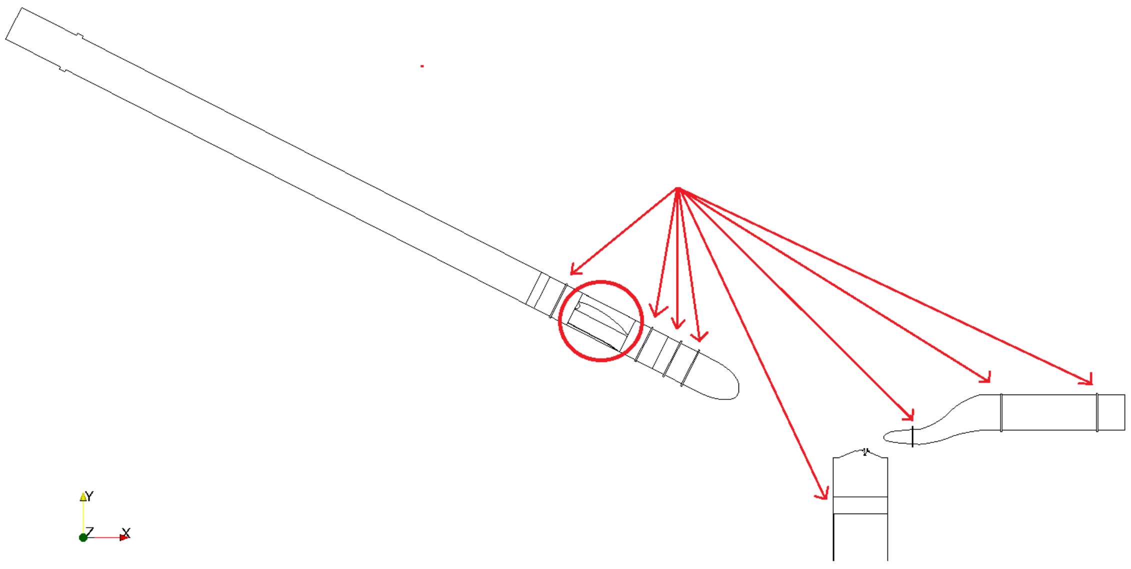


Figure 3.5: TU Darmstadt engine - slice, xy plane at $z = 0$ (symmetry plane). Errors highlighted.

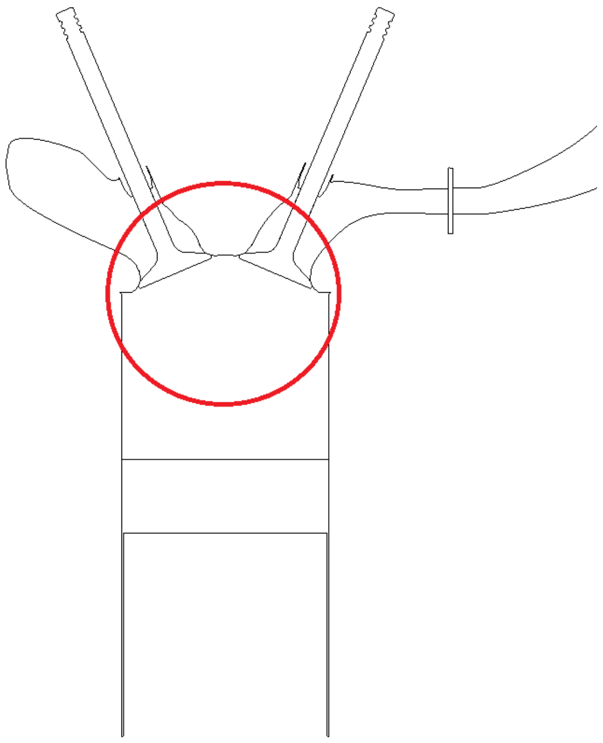


Figure 3.6: TU Darmstadt engine - slice, xy plane at $z = 0.02$. Errors highlighted.

From the first slice, shown in figure (3.5), it is possible to observe that the input geometry model is not suitable for CFD simulations. First of all there is an unknown surface inside the intake duct, highlighted by a red circle. This surface represents probably a measuring instrument (the location suggests the DN50 fused-silica cylinder of the experimental set-up), however it would strongly affect the flow field. Highlighted by red arrows through flanges are present in both the intake and exhaust ducts. These surfaces represent an undesired obstacle to the fluid motion.

A similar problem arises in the cylinder- and in the valve-regions, as shown in figure (3.6). The intake and exhaust valves are physically closed and this feature represents a problem for two utilities during the meshing step:

- the geometry does not enclose a unique volume. The *snappyHexMesh* utility requires an internal point in order to identify the volume to mesh: with more than one volume this means that only one sub-volume can be discretized (at least in OpenFOAM version preceding the last one -3.0.x).
- the valves motion would lead to distorted stl geometries. **createEngineSTL** utility moves the template stl geometry according to the prescribed motion

of valves and piston. The surfaces connected to these moving parts, move consequently: being the valves closed, they are connected to their seat on the cylinder head. The valve seats would thus follow the valve motion, while they are fixed components. For this reason the **engineDynamicSetUp** application requires a template stl file with all the valves open.

Analogous problem arises in the cylinder because of an attempt to design the combustion chamber crevices: the piston surface closes the geometry, thus the crevice height would not be part of the volume to discretize.

The first attempt to solve these problems has been made using MeshMixer. This software has a *inspector* tool able to identify the problems of the analysed geometry. Just to give to the reader a qualitative idea of the geometry quality, figure (3.7) shows the errors identified using the least strict control scale (thus neglecting a large number of other problems).

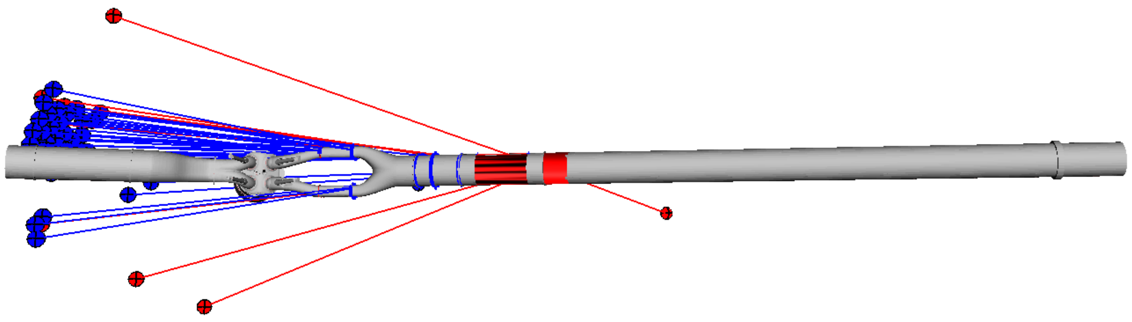


Figure 3.7: Geometry problems evaluated by the MeshMixer *inspector* tool.

The identified errors are mainly located in correspondence of the above mentioned improper internal surfaces.

In order to perform a correct CFD simulation using the assigned geometry, the improper internal surfaces inside the ducts have been removed. Furthermore, the intake ducts have been cut in correspondence of P_MAN2 (see figure (3.1)), since experimental data are available in this section.

Through MeshMixer, it was possible to singularly identify the geometry triangles using the *select* tool: in this way the geometry has been divided in a set of surfaces (the patches required by the meshing tool).

Paying attention to the surface triangulation, a first remark is that the geometry surface was not enough smooth, as it can be observed in figure (3.8). A poor smoothness can represent a problem for the utility *snappyHexMesh*.

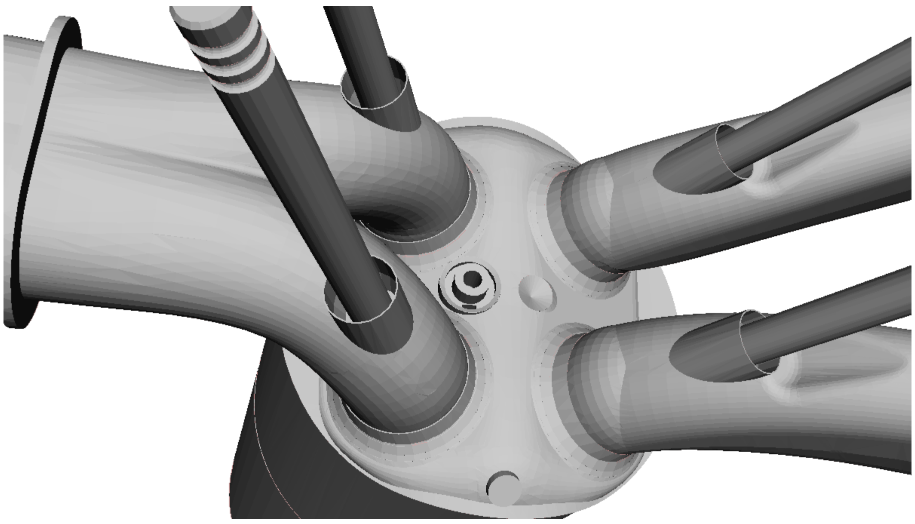


Figure 3.8: Zoom on the cylinder head: poor smoothness.

The *select* procedure of MeshMixer is shown in figure (3.9).

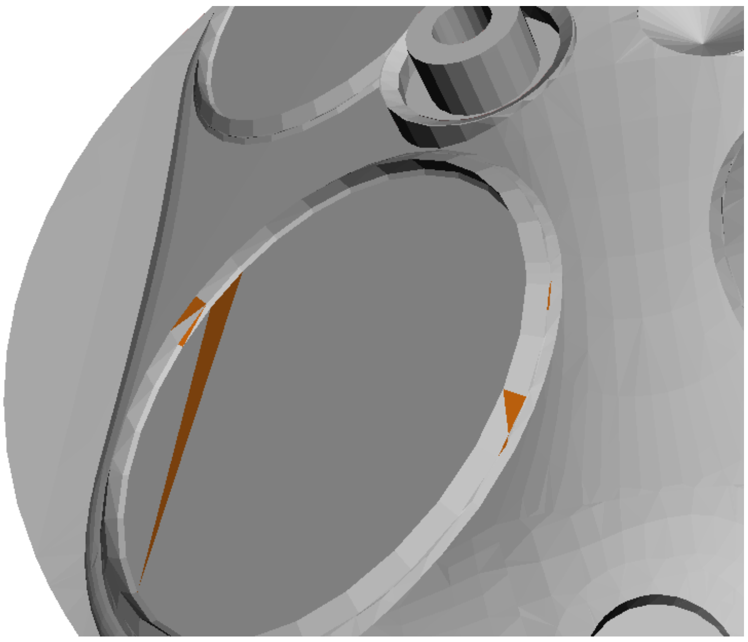


Figure 3.9: MeshMixer *select* tool: in orange the selected geometry triangles.

In this figure the exhaust ducts have been omitted to enable the examination of the contact between valve and its seat. The aim was to identify the triangles defining the valve components (specifically the bottom and side of the valve) and regroup them into two patches, separated from the patch defined by the triangles of the valve seat. This procedure encountered unexpected difficulties due to the irregular triangulation and to the negligible dimensions of some triangles to select.

Once separated these components, it was necessary to move the valves till the minimum lift. The valve motion uses the valve bottom as reference patch to move (other valve components move consequently). The motion however was not successful, due to the wrong orientation of the bottom surface normal, as shown in figure (3.10).

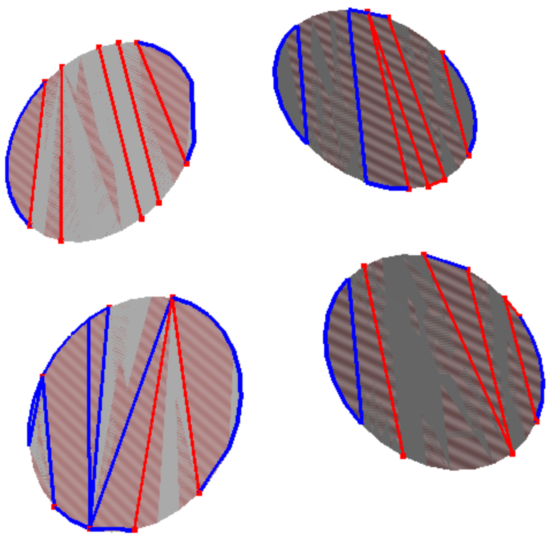
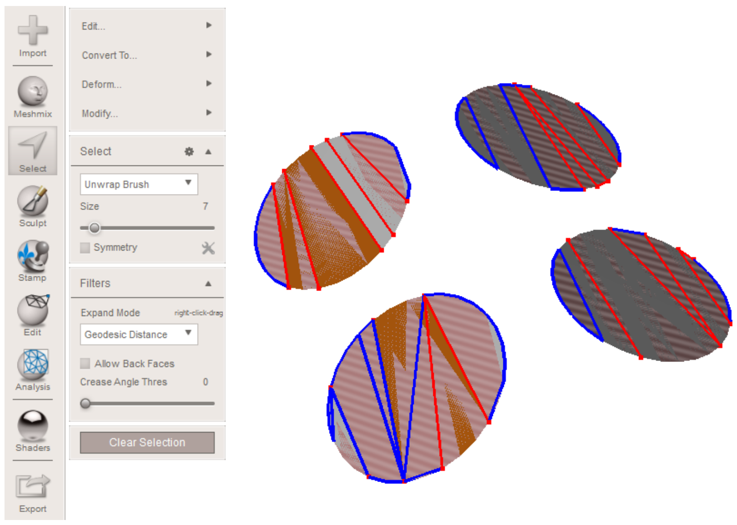


Figure 3.10: Valve bottoms patches: triangles with opposite surface orientation.

The triangles of the bottom valves had opposite surface orientation (identified by pink and grey colors): the motion solver was not able to identify an univocal surface orientation and thus a correct motion direction.

MeshMixer allows to invert the surface normal orientation, however selecting the wrong oriented triangles was a difficult task due to the triangles overlapping, as shown in figure (3.11).



Unwrap Brush
Size 7
Symmetry
Expand Mode right-click-drag
Geodesic Distance
Allow Back Faces
Crease Angle Thres 0

Figure 3.11: MeshMixer *select* tool: attempt to select the wrong oriented triangles.

Solving the valve closure issue led to a new problem: removing the contact between the valve and its seat, the surface of the whole geometry was open. In order to assure a closed surface, new external components (designed using the FreeCAD software) have been added to the given surface. Figure (3.12) shows the added component.

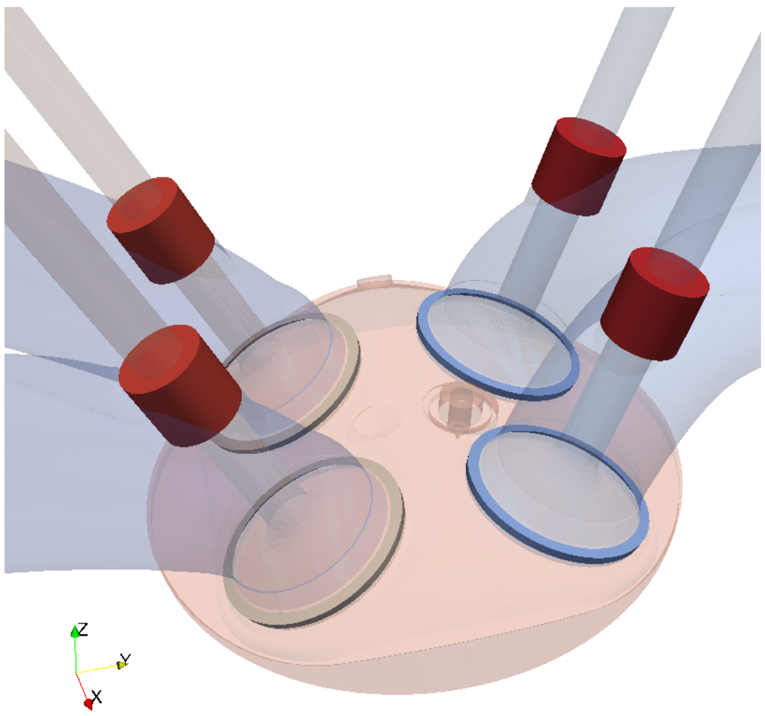
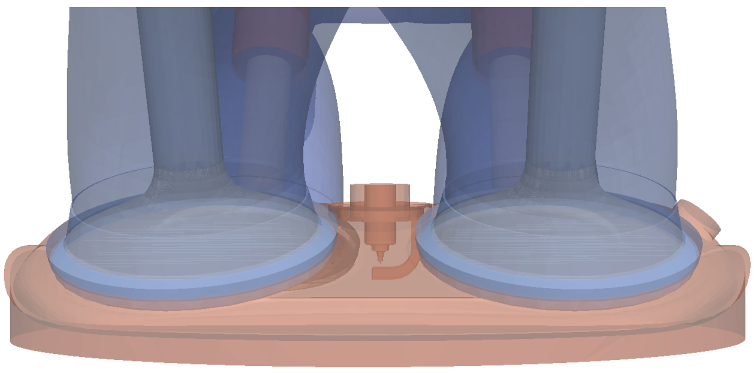


Figure 3.12: External components (full color) added to the given surface (transparent).

Finally, as the geometrical domain is symmetrical, the simulations have been performed on half geometry. However it was necessary to modify the spark-plug in order to ensure the exact geometrical symmetry (otherwise studying half geometry would not have been consistent). In figure (3.13), the original spark-plug and the spark-plug modified to ensure geometrical symmetry.

The modified geometry surface used as template file for the mesh generation is shown in figure (3.14).


(a) Original spark-plug.

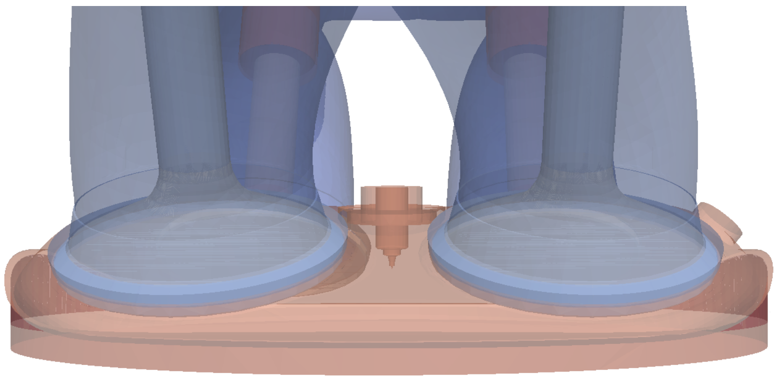

(b) Modified spark-plug.

Figure 3.13: Spark-plug modified to ensure geometrical symmetry.

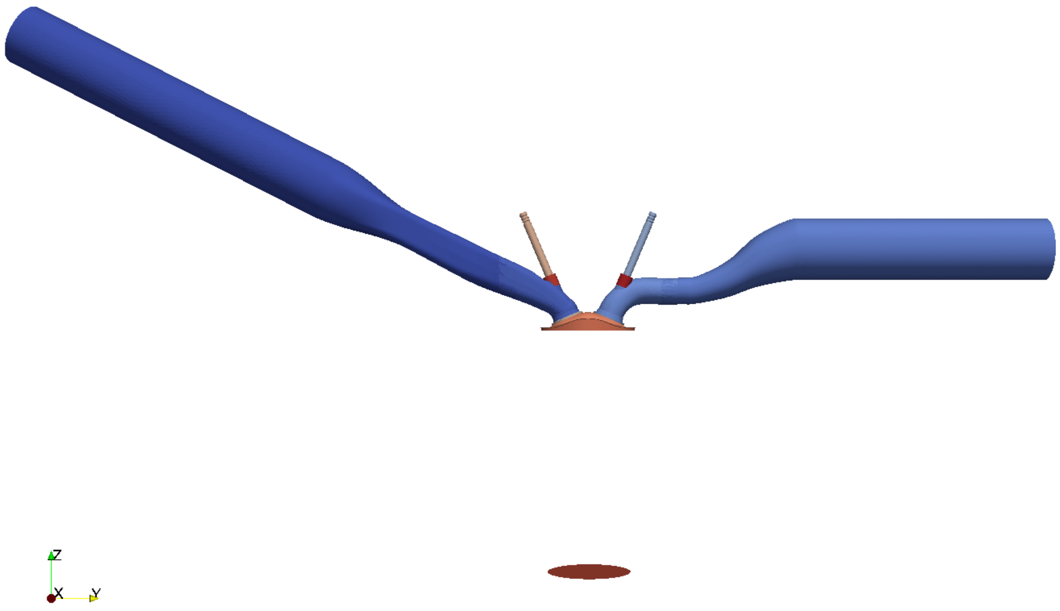


Figure 3.14: Template STL geometry.

3.2.2 Settings

The second requirement of **engineDynamicSetUp** is the set-up of all the dictionaries required in the different case folders, from those concerning the mesh generation, to those for the mesh motion till the ones requires for the following full-cycle simulations [10]. Figure 3.15 shows the structure of the case.

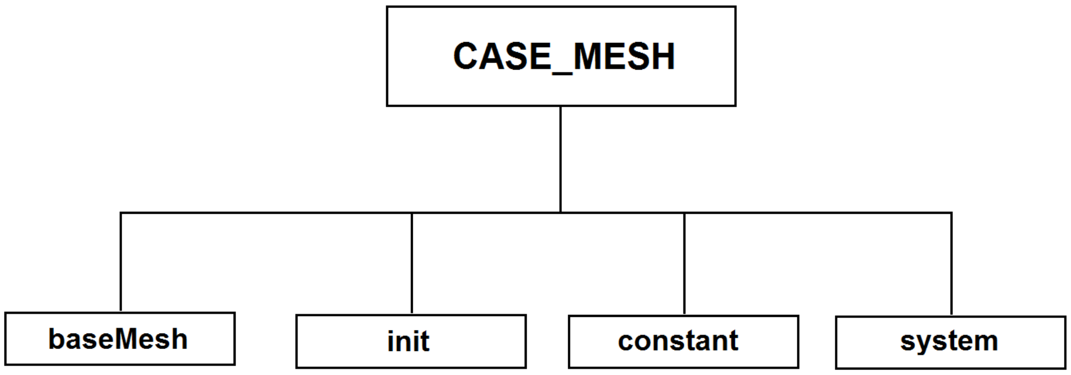


Figure 3.15: Case structure.

init folder: is the template directory that will be automatically copied to generate all the case folders required by the simulation. For this reason the set-up of the files in this folder should be done only once, before running the application.
As a "template" case, it has in turn three subfolders (figure (3.16)):

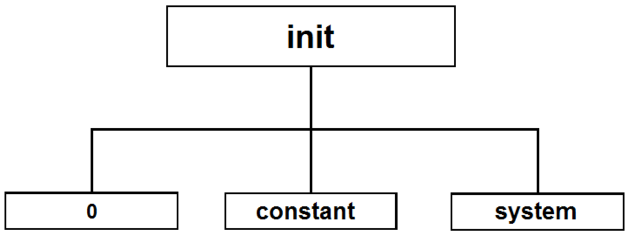


Figure 3.16: Structure of the init folder.

- **0**: it concerns initial and boundary conditions (fields). Specifically there are settings for chemical species (O2, CO2, H2O, N2, etc), thermodynamic quantities (p, T, U), turbulent quantities (k, epsilon, etc), wall film (filmFuel, hHfilm, Tfilm, etc) and a particular file, *cellMotionU* containing the motion boundary conditions for each patch. It has to be noticed that this file does not contain the right value of the boundary velocity for piston and valve patches: these values will be automatically calculated and updated by the solver according to the piston displacement and the valve lift profiles. In this dictionary are just specified the boundary condition types: fixed value for the patches that will not be moved and for those that will be moved with prescribed motion (piston and valveTop/Side/Bottom), zero gradient for the patches whose motion is calculated as a consequence of the prescribed boundary motion (liner and valveStem).

- **constant**: it concerns physical properties and geometry informations. Here are stored thermodynamic-chemistry-physical properties (chemistryProperties, combustionProperties, environmentalProperties, RASProperties, thermophysicalProperties, transportProperties, turbulenceProperties, etc), geometry informations in the *engineGeometry* dictionary, valve lift profiles in *exhaustValveLift.txt* and *intakeValveLift.txt* and finally the dictionary *dynamicMeshDict*, which handles the mesh motion method, in its version *inversePD* and *inverseVolume*: during the mesh motion step, both the inverse point distance and inverse volume methods (explained in section 2.1.1) are tested.

- **system**: it concerns simulation control parameters and numerics. Specifically, in addition to the traditional run-time, numerical schemes and solution settings of the simulation (controlDict, fvSchemes, fvSolution), there are dictionaries for the fields control (mapFieldsDict, setFieldsDict) and the dictionary for the parallel decomposition (decomposeParDict): for a full-cycle simulation, the possibility to run it in parallel is crucial.

constant folder: it concerns geometry information. In this folder are located again *engineGeometry* dictionary and valve lift profiles. The *engineGeometry* is the same stored in init/constant folder: here geometric parameters like bore, stroke, clearance, etc are reported together with information necessary for the following motion: the chosen mesh motion strategy, the coordinate system relative to the patches to move (used to identify the translation axis), the minimum lift below which valves are considered closed, quality parameters for the motion stage and finally some post-processing information data.

For what concerns the valve lift profiles, the files located in this folder (*exhaust-ValveLift* and *intakeValveLift*) differ from those located in the init/constant folder (*exhaustValveLift.txt* and *intakeValveLift.txt*): these files are used to impose the valve movement at the stl template geometry (which is at TDC with each valve at the minimum lift), therefore the values contained in these files are shifted by the value of the minimum lift.

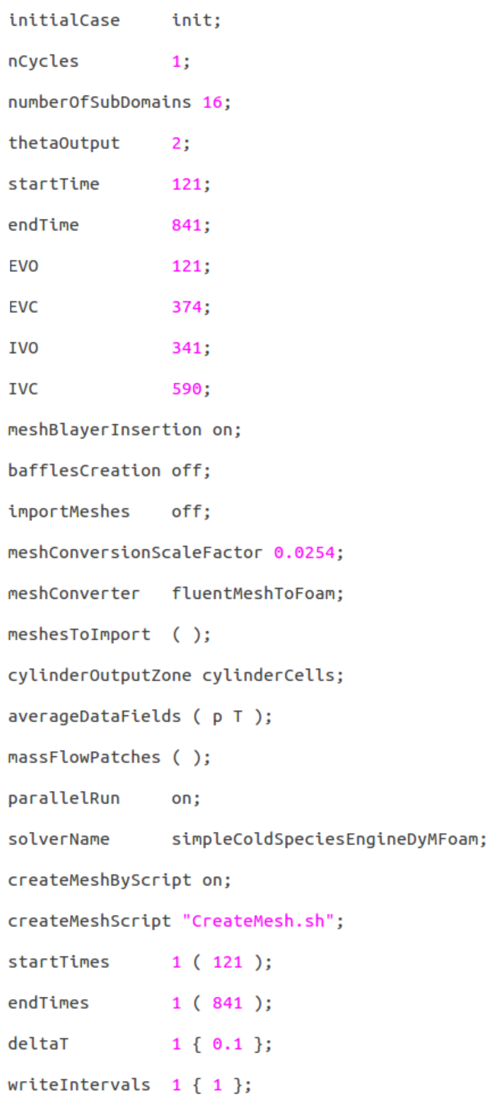
In addition to those files, here is stored also the *injectorProperties* dictionary, in case of a direct injection spark-ignition engine.

system folder: as it is the global system folder, here (among other typical dictionaries like decomposeParDict, changeDictionaryDict, controDict, etc.) is located the dictionary responsible for the actual management of the **engineDynamicSetUp** application and later of the **runMultiCycleCase** application: *engineControlDict*. As each -ControlDict, it contains settings like start- and end-time of the simulation (here expressed in crank-angle degrees), post-processing data and solver name (of the following cold-flow simulation). However this dictionary includes additional statements specific for a full-cycle simulation: information about the valve timing, that is exhaust valve opening (EVO), exhaust valve closing (EVC), intake valve opening (IVO) and intake valve closing (IVC), each in crank-angle degrees. These information are necessary to automatically identify the engine phase. Each phase requires specific setting during the mesh generation.

Moreover it is possible to specify how the single-mesh generation will be handled: it is possible to customize the mesh generation by writing a script containing all the desired steps of the generation, instead of using the standard one. The forementioned script must be specified under the statement *createMeshScript*.

Finally a list of *startTimes*, *endTimes*, *deltaT* and *writeIntervals* ends the dictionary. At the beginning these statements contains only the values already specified as start- and end-time, deltaT (time step for the mesh motion) and writeInterval (interval by which record the simulation results). However, during the mesh generation, these fields will be updated with the end-time of the mesh validity, corresponding to the start-time of the next one. The updated dictionary will be used for the following cold-flow simulation (after the user-choice of deltaT and writeInterval, as will be

explained in section 3.3). Since the *engineControlDict* dictionary will be updated during the generation of the full-cycle set of mesh, it is necessary to copy it in a backup file before running the **engineDynamicSetUp** application.
Figure 3.17 shows the *engineControlDict* statements.



```
initialCase     init;
nCycles         1;
numberOfSubDomains 16;
thetaOutput     2;
startTime       121;
endTime         841;
EVO             121;
EVC             374;
IVO             341;
IVC             590;
meshBlayerInsertion on;
bafflesCreation off;
importMeshes    off;
meshConversionScaleFactor 0.0254;
meshConverter   fluentMeshToFoam;
meshesToImport  ( );
cylinderOutputZone cylinderCells;
averageDataFields ( p T );
massFlowPatches ( );
parallelRun     on;
solverName      simpleColdSpeciesEngineDyMFoam;
createMeshByScript on;
createMeshScript "CreateMesh.sh";
startTimes      1 ( 121 );
endTimes        1 ( 841 );
deltaT          1 { 0.1 };
writeIntervals  1 { 1 };
```

Figure 3.17: engineControlDict.

baseMesh folder: it is used for the mesh generation. As shown in figure (3.18), it has in turn three subfolders:

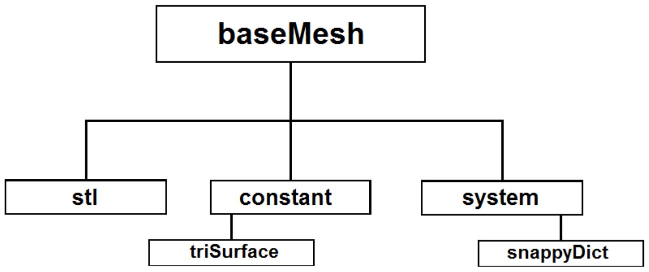


Figure 3.18: Structure of the baseMesh folder.

- **stl**: at the beginning this folder is empty. Here will be stored the stl files generated by the utility **createEngineSTL**. This utility creates, starting from the template stl file, new stl geometry at the last-valid (corresponding to the starting new) crank-angle. These files are required by the mesh generator (*snappyHexMesh*) to make the meshes at different crank-angles.

- **constant**: as for the other constant folders, it concerns geometry information. There is the *triSurface* subfolder, where all the stl files required during the mesh generation are stored (e.g. template.stl which is the template geometry file used by **createEngineSTL** utility, geometry.stl which is the geometry file used by *snappyHexMesh*, boxIn.stl and boxEx.stl used to define cell zones, etc.).

  If the OpenFOAM version used to generate the mesh is older than the last one (-3.0.x), it is required also the *polyMesh* subfolder, where the *blockMeshDict* dictionary, used to generate the background grid with **blockMesh**, is stored. Once the mesh is generated, in this folder will be stored the files defining the grid (points, faces, owner, neighbour, boundary, pointZones, faceZones). On the contrary, by using the last OpenFOAM version, the *blockMeshDict* dictionary is stored in the *system* folder: the *polyMesh* subfolder will be automatically generated after the mesh generation, its initialisation is not required.

- **system**: in this folder are located all the dictionaries which rule the stl creation and the mesh generation and manipulation.
  The generation of the required stl geometry files is controlled by the *createEngineSTLDict*. Here the user must specify the name of the template stl file (located in constant/triSurface) and the list of crank-angles at which stl geometry files are required. This list can be imposed by the user a priori or

calculated run-time by the **engineDynamicSetUp** application. In the former case, the prescribed list must be written in the dictionary itself. In the latter case, the application automatically creates a file (moveOutput.txt) containing the last valid crank-angle position of the moved mesh (coincident with the starting position of the next mesh): this file will be used to specify the required crank-angle.
Concerning the mesh generation, the *snappyHexMesh* utility is ruled by the *snappyHexMeshDict* dictionary. During a full-cycle simulation, the engine phase changes: each phase has different mesh requirements. For this reason it is necessary to have a different dictionary for each phase and for the layer insertion: snappyDict_compression_layer, snappyDict_compression_snap, snappyDict_exhaust_layer, snappyDict_exhaust_snap, snappyDict_intake_layer, snappyDict_intake_snap, snappyDict_overlap_layer, snappyDict_overlap_snap. These dictionaries are collected in the *snappyDict* subfolder. The application will recognize the current engine phase and use the correct dictionary for the *snappyHexMesh* utility.
Finally the mesh can be manipulated by the utilities *createPatch*, *topoSet*, *createBaffles* and *extrudeMesh*. The *topoSetDict* governing the *topoSet* utility allows to create the set of cells required for further manipulation (createBaffles, extrudeMesh) and for post-processing analyses. The *createBaffles* utility, ruled by the createBafflesDict, is used to close the valves by adding set of cells in correspondence of the gap due to the minLift. Similarly to the *snappyHexMeshDict*, also the *createBafflesDict* must change according to the engine phase. Finally the *extrudeMesh* utility is used to generate the combustion chamber crevices, whose height increases with the piston moving towards the TDC.

The application automatically creates a case folder for each mesh generated, where the init folder is copied together with the polyMesh subfolder defining the mesh itself. At the end of the generation of the set of mesh, the case structure appears as in figure (3.19).

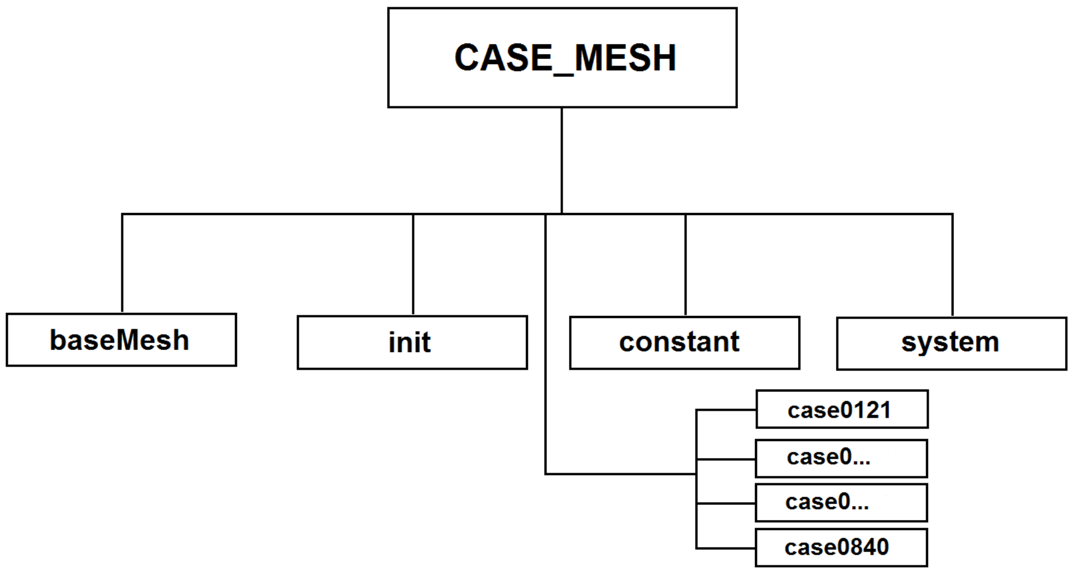


Figure 3.19: Final structure of the case.

3.3 runMultiCycleCase

The full-cycle gas exchange simulation has been performed with another fully automatic utility implemented in the Lib-ICE library, **runMultiCycleCase**. It is an application for the management of a full-cycle cold-flow simulation, perfectly coupled with **engineDynamicSetUp** application.

Once the full-cycle set of meshes has been generated and the mesh case is complete (figure (3.19)), the **runMultiCycleCase** takes it as input case. To work properly it requires only 2 additional steps:

- the addition of the **dataTime** folder
- the proper setting of *deltaT* and *writeIntervals* statements in *engineControlDict*

Figure 3.20 shows the structure of the case.

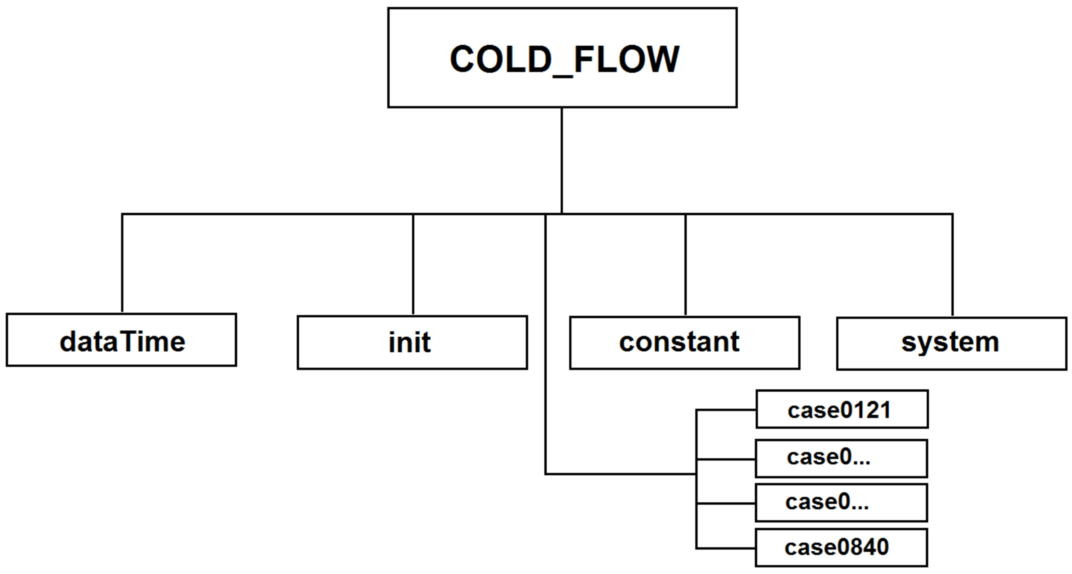


Figure 3.20: Case structure for the cold-flow simulation.

In the **dataTime** folder are stored the input files for the simulation, containing boundary conditions for the flow field: *p_exh1.t* and *p_man2.t* files. These files consist in two column text with crank angles (first column) and corresponding pressure values (second column) at the inlet (p_man2) and outlet (p_exh1) of the engine. In order to have a correct interpolation on the entire time domain (an entire engine cycle), data must be specified from CA 0 to 720, including both. In the present work the boundary conditions are the experimental data provided by TU-Darmstadt. Figure (3.21) shows the unsteady pressure conditions imposed at inlet and outlet ports.

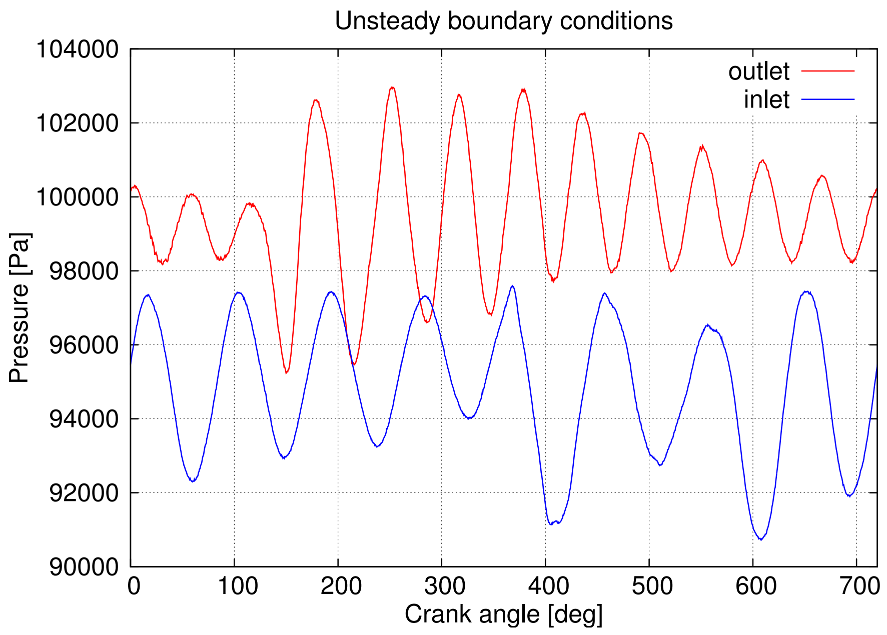


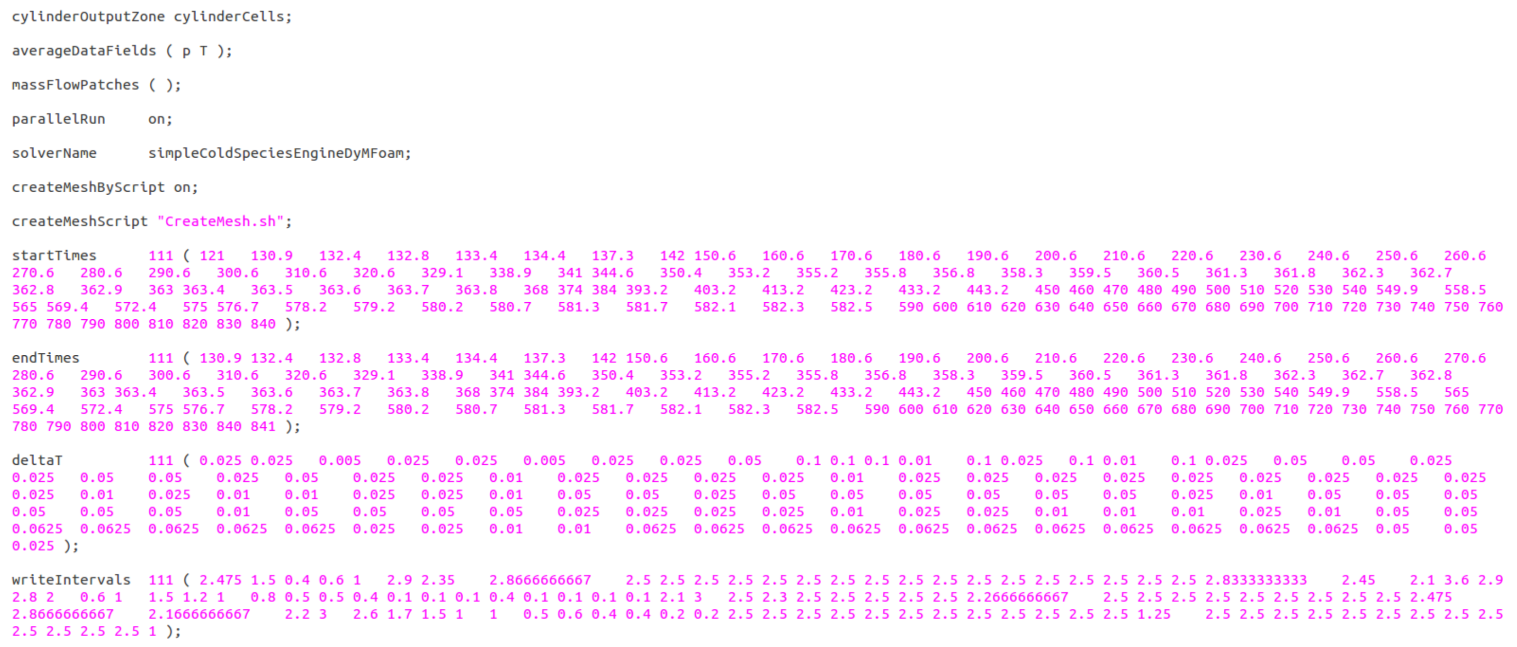
Figure 3.21: Unsteady pressure conditions imposed at the boundaries through the files *p_ exh1.t* and *p_ man2.t*.

Concerning the *engineControlDict*, as previously explained the startTimes and endTimes fields are updated according to the mesh generation and motion, while the deltaT and writeIntervals fields must be filled in by the user:

- *deltaT*: here are listed the simulation time-steps for each case. The choice of the time-step should be a compromise between high stability and manageable computation costs. Lower time step values are required at initial time and at the beginning of a new engine phase to increase stability. Moreover these values must be sub-multiples of the corresponding case duration. In the present work, the time-step values go from a minimum of 0.01 to a maximum of 0.0625.

- *writeIntervals*: here are listed the write-intervals for each case. These values represent the CA interval after which write results. A standard approach is to have from two to four records for each case. The interval must be a multiple of the time-step of the case and a sub-multiple of the case duration.

Figure (3.22) shows the final part of *engineControlDict* setted for the gas exchange simulation.



```
cylinderOutputZone cylinderCells;

averageDataFields ( p T );

massFlowPatches ( );

parallelRun     on;

solverName      simpleColdSpeciesEngineDyMFoam;

createMeshByScript on;

createMeshScript "CreateMesh.sh";

startTimes      111 ( 121   130.9   132.4   132.8   133.4   134.4   137.3   142 150.6   160.6   170.6   180.6   190.6   200.6   210.6   220.6   230.6   240.6   250.6   260.6
270.6   280.6   290.6   300.6   310.6   320.6   329.1   338.9   341 344.6   350.4   353.2   355.2   355.8   356.8   358.3   359.5   360.5   361.3   361.8   362.3   362.7
362.8   362.9   363 363.4   363.5   363.6   363.7   363.8   368 374 384 393.2   403.2   413.2   423.2   433.2   443.2   450 460 470 480 490 500 510 520 530 540 549.9   558.5
565 569.4   572.4   575 576.7   578.2   579.2   580.2   580.7   581.3   581.7   582.1   582.3   582.5   590 600 610 620 630 640 650 660 670 680 690 700 710 720 730 740 750 760
770 780 790 800 810 820 830 840 );

endTimes        111 ( 130.9 132.4   132.8   133.4   134.4   137.3   142 150.6   160.6   170.6   180.6   190.6   200.6   210.6   220.6   230.6   240.6   250.6   260.6   270.6
280.6   290.6   300.6   310.6   320.6   329.1   338.9   341 344.6   350.4   353.2   355.2   355.8   356.8   358.3   359.5   360.5   361.3   361.8   362.3   362.7   362.8
362.9   363 363.4   363.5   363.6   363.7   363.8   368 374 384 393.2   403.2   413.2   423.2   433.2   443.2   450 460 470 480 490 500 510 520 530 540 549.9   558.5   565
569.4   572.4   575 576.7   578.2   579.2   580.2   580.7   581.3   581.7   582.1   582.3   582.5   590 600 610 620 630 640 650 660 670 680 690 700 710 720 730 740 750 760 770
780 790 800 810 820 830 840 841 );

deltaT          111 ( 0.025 0.025   0.005   0.025   0.025   0.005   0.025   0.025   0.05    0.1 0.1 0.1 0.01    0.1 0.025   0.1 0.01    0.1 0.025   0.05    0.05    0.025
0.025   0.05    0.05    0.025   0.05    0.025   0.025   0.01    0.025   0.025   0.025   0.025   0.01    0.025   0.025   0.025   0.025   0.025   0.025   0.025   0.025   0.025
0.025   0.01    0.025   0.01    0.01    0.025   0.025   0.01    0.05    0.05    0.025   0.05    0.05    0.05    0.05    0.05    0.05    0.025   0.01    0.05    0.05    0.05
0.05    0.05    0.05    0.01    0.05    0.05    0.05    0.05    0.025   0.025   0.025   0.025   0.01    0.025   0.025   0.01    0.01    0.01    0.025   0.01    0.05    0.05
0.0625  0.0625  0.0625  0.0625  0.0625  0.025   0.025   0.01    0.01    0.0625  0.0625  0.0625  0.0625  0.0625  0.0625  0.0625  0.0625  0.0625  0.0625  0.0625  0.05    0.05
0.025 );

writeIntervals  111 ( 2.475 1.5 0.4 0.6 1   2.9 2.35    2.8666666667    2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.8333333333    2.45    2.1 3.6 2.9
2.8 2   0.6 1   1.5 1.2 1   0.8 0.5 0.5 0.4 0.1 0.1 0.1 0.4 0.1 0.1 0.1 0.1 2.1 3   2.5 2.3 2.5 2.5 2.5 2.5 2.5 2.2666666667    2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.475
2.8666666667    2.1666666667    2.2 3   2.6 1.7 1.5 1   1   0.5 0.6 0.4 0.4 0.2 0.2 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 1.25    2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5
2.5 2.5 2.5 2.5 1 );
```

Figure 3.22: engineControlDict -final part.

Chapter 4

Mesh generation

In this chapter the mesh generation for the case under analysis is described. In the first section the choice of grid type is explained, together with the consequent background grid generation strategy. In the second part of the chapter the customized single-mesh generation procedure is explained, focusing on its main step, the generation of the body-fitted mesh.

4.1 Background grid

The background grid is a requirement of *snappyHexMesh*: it is a mesh of hexahedral cells that fills the entire region within by the external boundary, defining the extent of the computational domain and a base level mesh density.

Taking into account the previous work [5] of the ICE Group of Politecnico di Milano, it has been chosen to generate a flow-oriented grid instead of a fully cartesian one: in a cartesian grid the cell edges have the same directions of the cartesian coordinate system, while in a flow oriented grid the cell edges directions comply with the body directions. In a cartesian mesh numerical diffusivity spreads the entering flow over the grid, smoothing the velocity field gradients, while using a flow-oriented grid the effects of the artificial diffusivity are reduced.

This conclusion comes from different sensitivity analysis on flow-grid alignment. It is possible to define a flow grid alignment index (FGA) describing how the flow is aligned with the mesh:

$$FGA = \frac{\sum_f \left(\rho S U_N \arctan\left(\frac{U_N}{U_T}\right)\right)}{\frac{\pi}{2}\sum_f \left(\rho S U_N\right)} \qquad 0.5 \leq FGA \leq 1 \tag{4.1.1}$$

where $FGA = 1$ identifies a perfectly flow oriented grid and $FGA = 0.5$ identifies a grid having an angle of 45° between the flow direction and the face normal. FGA does not represent a grid quality index, but a quality index for the CFD simulation.

The results of the sensitivity analysis on flow-grid alignment applied to a laminar jet are shown in figure (4.1).

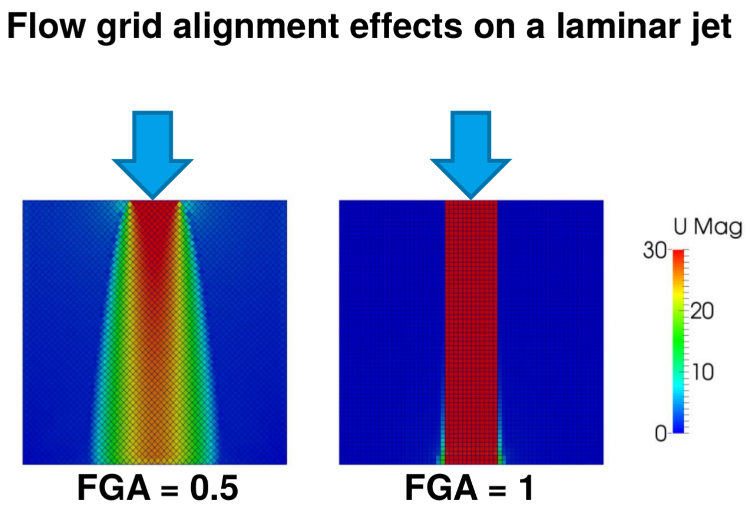


Figure 4.1: Flow grid alignment effects on a laminar jet.

Same conclusion has been obtained by the ICE Group during the sensitivity analysis on flow-grid alignment applied to a gas exchange simulation on the standard configuration of the Darmstadt engine.

Starting from these considerations it has been chosen to generate a flow-oriented grid for the present work. The generation of a flow-oriented grid requires greater effort than the cartesian one: the background grid generated by *blockMesh* must be block-structured, while for a cartesian grid a unique block is sufficient. Figures (4.2) and (4.3) show the multiple blocks composing the structured background grid.

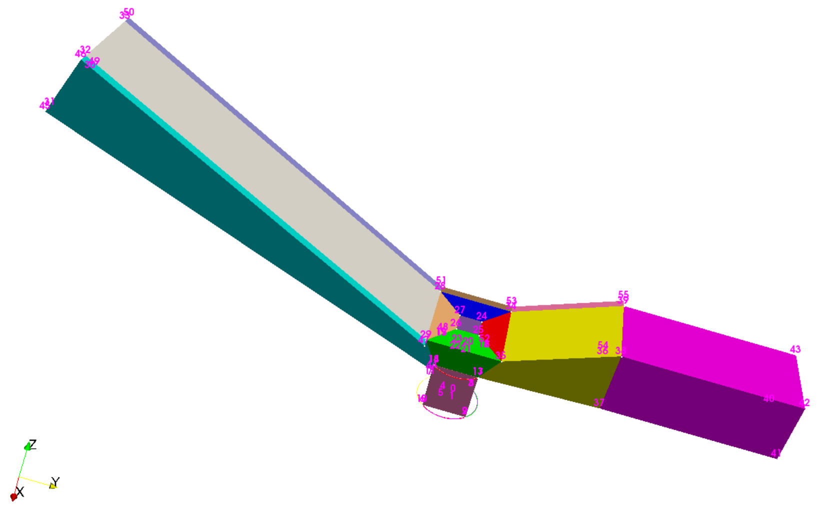


Figure 4.2: Multiple oriented blocks composing the structured background grid.

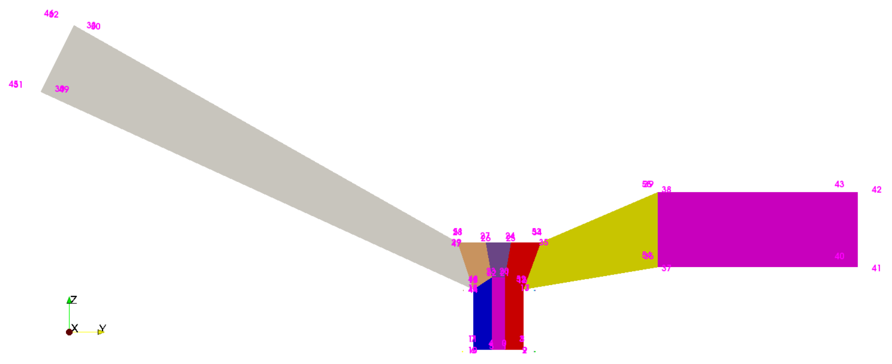


Figure 4.3: Multiple oriented blocks composing the structured background grid - slice in correspondence of the symmetry plane.

In order to handle the moving geometry, the background grid has to move in compliance with it: in the **baseMesh** folder (see section 3.2.2) has been added the subfolder **blockMeshGeneration**, where python scripts able to dynamically manage the background grid are stored.

First of all the python script *modifyDicts.py* reads the geometry file geometry.stl stored in baseMesh/trisurface. This file is the temporary copy of the stl file generated by **createEngineSTL**, updated at each new stl -and thus mesh- generation. According to the current piston displacement, this script changes:

- the python script *controlPanel.py*, containing reference values necessary to govern the background grid generation. This script is read by the *blockMesh.py* script which actually generate the *blockMeshDict* ruling the *blockMesh* utility. In particular *modifyDicts.py* modifies the values of *stroke* and *nZ*, two statements ruling the height of the cylinder block and the number of its cells in the vertical direction respectively. As shown in figure (4.4), the height of the cylinder block is equal to the stroke of the current geometry.

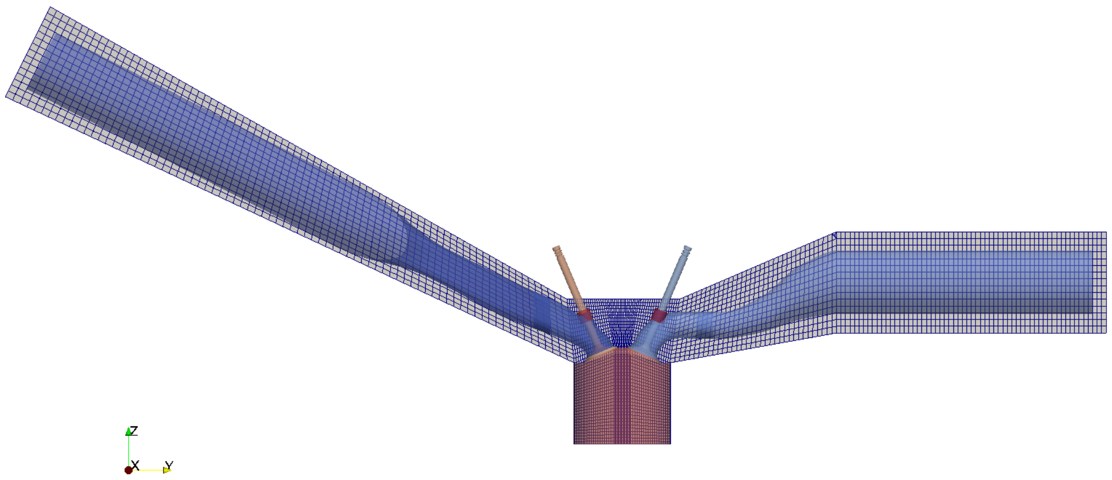


Figure 4.4: Background grid: perfect matching between current stl geometry and mesh.

  The number of cells in vertical direction changes with the stroke, in order to guarantee a constant grid resolution.

- the *extrudeMeshDict* dictionary, located in baseMesh/system. This dictionary rules the *extrudeMesh* utility, responsible for the combustion chamber crevices extrusion. In particular *modifyDicts.py* modifies the value of *thickness*, ruling the height of the crevices according to the piston position.

The dynamical generation of the *blockMeshDict* has two objectives: maintaining a constant mesh resolution and easily handling the crevice extrusion. Being the cylinder block perfectly coincident with the cylinder of the geometry, it is possible to re-define inside the *blockMeshDict* the patch *piston* in order to introduce the patch *crevice*: an annulus as thick as the combustion chamber crevices (0,5 mm). In figure (4.8) it is possible to observe the external annulus representing the crevice patch. This patch will be then extruded according to *extrudeMeshDict*, thus generating the crevices.

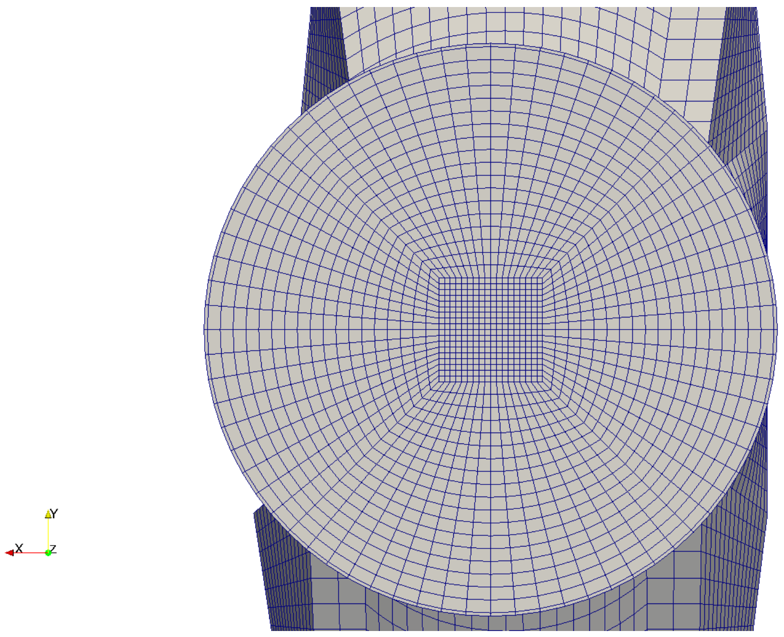


Figure 4.5: Piston and crevice patches in the background grid.

The grid size varies in function of the geometry regions:

- 4 mm mesh size in the ducts region
- 1 mm mesh size in the cylinder and valve region

snappyHexMesh utility applies further refinement in the cylinder and valve regions.

The choice of the grid size has been made taking into account similar analyses [5], with finer resolution in the cylinder-valve region in order to allow an eventual combustion process analysis (in keeping with [25]).

4.2 Customized mesh generation procedure

The single-mesh generation has been handled using a script, *CreateMesh.sh*, stored in the baseMesh folder. First it runs the python scripts *modifyDicts.py* and *blockMesh.py*, thus generating the proper *blockMeshDict*. Then it executes the actual mesh generation:

- background grid generation: **blockMesh** utility
- selection of one-half of the whole grid domain (thus exploiting the geometrical symmetry to reduce computational costs): **topoSet**, **subsetMesh** and **createPatch** utilities

- selection of the proper *snappyHexMeshDict* according to the the current engine phase
- decomposition of the computational domain and generation of the body-fitted mesh (parallel running [1]): **decomposePar** and **snappyHexMesh** utilities
- addition of the boundary layer (according to the current engine phase): **snappyHexMesh** utility
- reconstruction of the the domain: **reconstructParMesh** utility
- generation of the baffles closing the valve (according to the current engine phase): **topoSet**, **createPatch** and **createBaffles** utilities
- extrusion of the combustion chamber crevices: **extrudeMesh**, **topoSet** and **createPatch** utilities

The main step of the procedure consists in the generation of the body-fitted mesh by means of **snappyHexMesh**.

4.2.1 snappyHexMesh

Particular attention has been paid to the definition of the different versions of the *snappyHexMeshDict*. It was necessary to rely on the functionalities of *snappyHexMesh* to overcome the poor quality of the geometry. In particular, when the valves are close to the minimum lift, the gap between valve and valve seat is so small that introducing cells in-between is critical. For this reason, it has been used the OpenFOAM release -3.0.x for the generation of the body-fitted mesh. Indeed in this OpenFOAM version new meshing functionalities have been implemented, as the ability to detect and refine regions with close features automatically, e.g. gaps [20]. Thus a high surface refinement level has been applied to valve patches (to the valve diagonal and the valve seat in particular), together with the above mentioned gap refinement. In figure (4.6) it is possible to observe the high refinement applied in the valve region.

[1]the domain has been divided in 16 sub-domains. The mesh-generation simulation as well as the following cold-flow simulation have been run in parallel on the CINECA GALILEO platform.

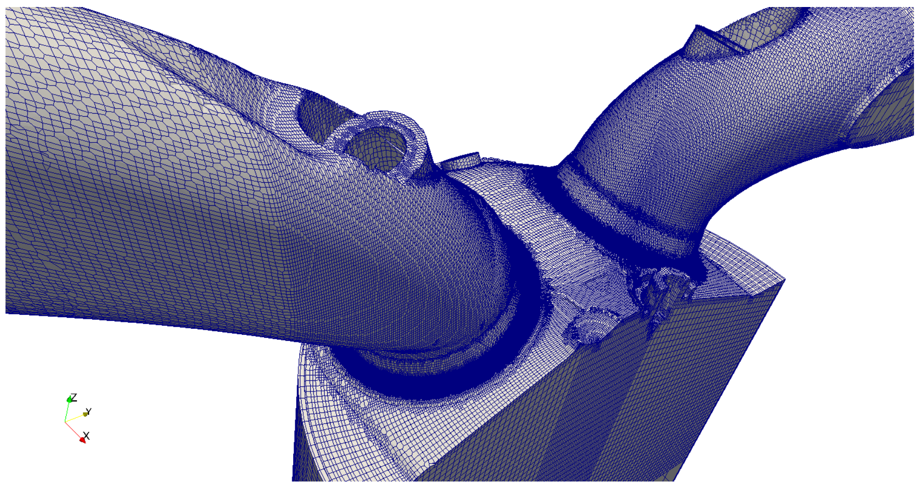


Figure 4.6: Refined grid on the cylinder-head and valve region.

Moreover this figure shows the further refinement regions applied, in correspondence on the intake and exhaust valve region (including the final parts of the ducts) and of the cylinder region.

As already explained, each engine phase has its peculiarities, with consequent grid requirements and complications. Except for the intake and exhaust ducts, the geometrical domain changes: the internal grid must accommodate the boundary motion still preserving the mesh quality.

For each phase is thus necessary to focus on the *meshQualityControl* settings. The critical meshing phase occurs when the piston approaches the TDC, that is during the overlap and the compression phases. In this phases the in-cylinder volume is minimum and it is difficult to maintain a certain mesh resolution without incurring in degenerated cells. The grid must be fine in order to ensure reliable results, above all during the compression phase: if the grid must be suitable for the combustion process analysis, a small grid size is an essential requirement for a correct prediction of the ignition and the following combustion evolution [23, 25]. However it is a difficult task to maintain a proper number of cell in the vertical direction without generating improper cells. In correspondence of the contact between liner and cylinder-head the vertical available space is so narrow that the non-orthogonality and the minimum volume of the cells are quality parameters difficult to fulfill. For this reason the *snappyHexMeshDict* differs in these quality settings according to the engine phase: the *non-orthogonality* (maxNonOrtho) varies within 50 and 60 and the *minimum volume* (minVol) within 1e-16 and 1e-15. Too strict quality requirements lead to the impossibility to generate the body-fitted mesh, while too weak requirements lead to degenerated cells (wrong oriented, non-orthogonal, skew, etc.). Figure (4.7) shows the grid at the top-dead center.

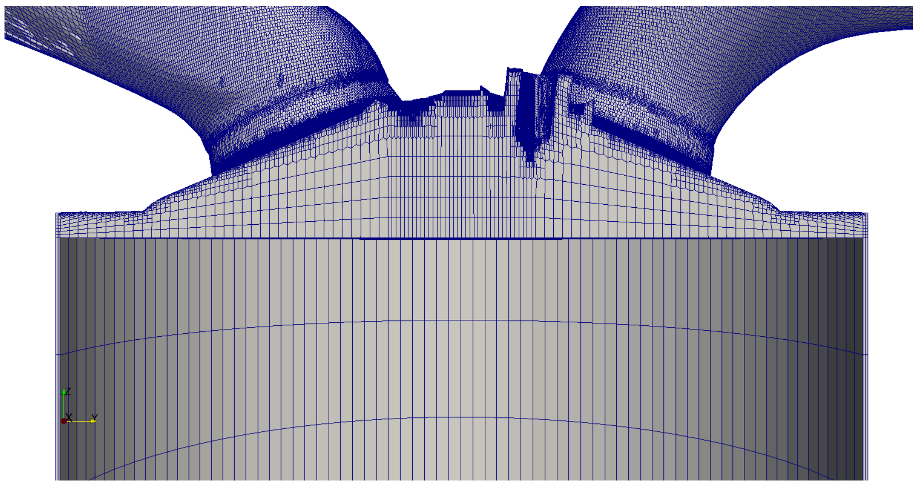


Figure 4.7: Mesh at the TDC.

The mesh generation during the overlap phase is the most complex: the piston is in correspondence of TDC and furthermore the valves are open with a lift almost at the minimum value. In this phase the two main problems of the meshing phase (piston at TDC and small gap between valve and its seat) coexist.

The awarded validity interval of the mesh during the following motion step reveals the encountered difficulties during the mesh generation: during the overlap phase and at the beginning of a new phase (that is when the valve lift is close to the minLift), the CA-interval where the mesh preserves its validity decreases remarkably, up to the minimum range of 0.1 CA. During most of the other phases instead the mesh preserves its validity up to the maximum interval imposed, that is 10 CA.

Finally, once generated the body-fitted mesh, it is possible to add the boundary layers. Even the layering phase is handled by *snappyHexMeshDict*, however the quality parameters must be softened in order to achieve the layer generation. The layer addition in fact involves shrinking the existing mesh from the boundary and inserting layers of cells, procedure affecting the mesh quality.

For this reason the generation of the mesh is governed by two versions of *snappyHexMeshDict* per engine phase: *snappyHexMeshDict_ snap* and *snappyHexMeshDict_ layer* (see section 3.2.2).

The differences of the *snappyHexMeshDict_ layer* with respect to the *snappyHexMeshDict_ snap* version consist in softened validation criteria, in particular the maximum allowed value concerning the non-orthogonality check is 70 and the minimum tet quality check is disabled.

The differences between the four *snappyHexMeshDict_ layer* (one per engine phase) instead, concern the patches on which the layers are to be applied. Specifically, during the intake phase layers must be applied on the intake walls, the intake valve components and on the cylinder-head (as the exhaust-side is closed), analogously during the exhaust phase layers must be applied on the exhaust walls, the exhaust valve components and on the cylinder-head. During the overlap phase both the intake and exhaust side are involved, therefore layers must be applied on the intake and exhaust walls, the intake and exhaust valve components and on the cylinder-head. On the contrary during the compression phase both the intake and exhaust are closed and the layers must be applied only on the cylinder-head.

It has to be remarked that no layers are applied on the liner. Usually layers are required on this patch, however in the present case their addition is not necessary: in the background grid the blocks are defined so that an external annulus covers the entire cylinder stroke. This annulus is required for the crevices extrusion, but at the same time represents a sort of layer. A further layers addition is thus not necessary. Figure (4.8) gives a visual explanation of this concept.

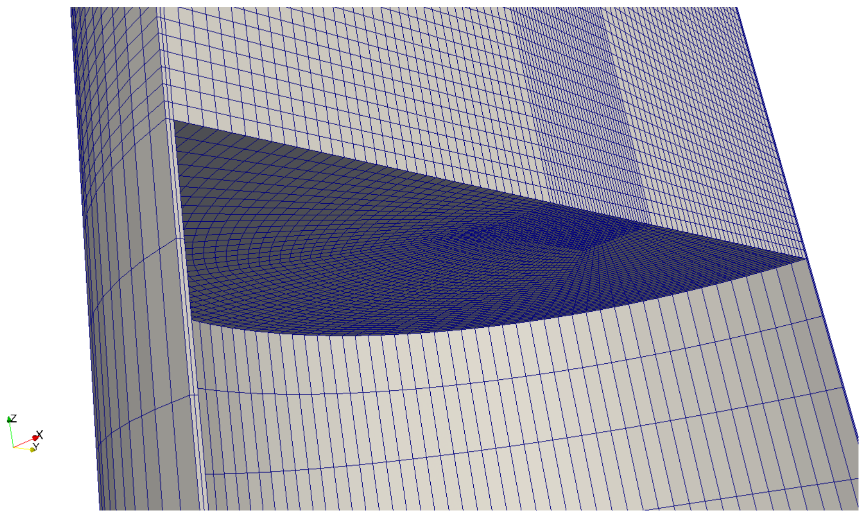


Figure 4.8: Zoom on the piston and liner patches and the already extruded crevices.

Figures (4.9) and (4.10) show the oriented grid in the intake and exhaust ducts respectively. These zones are the less complicated since they are not affected by motion: particular attention must be paid only during the background grid generation in order to achieve a good flow-grid alignment.

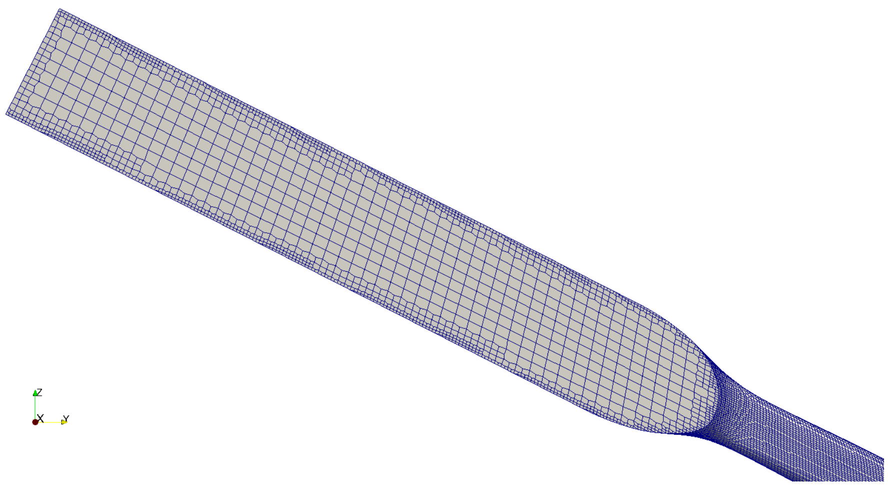


Figure 4.9: Oriented grid in the intake duct.

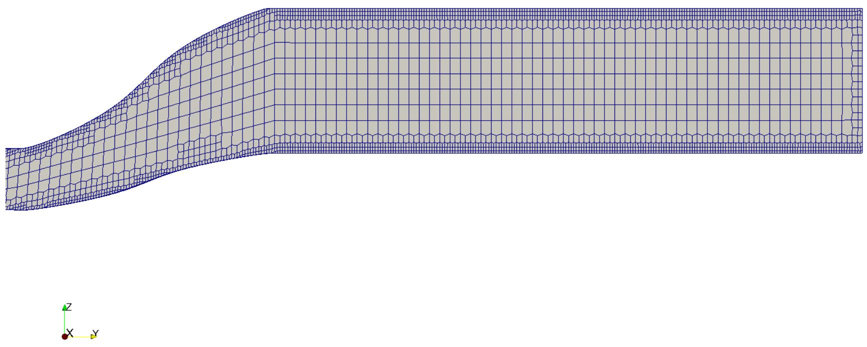


Figure 4.10: Oriented grid in the exhaust duct.

Figure (4.11) shows the achieved in-cylinder mesh and part of the extruded crevices. Further region refinements have been applied in the cylinder region and around the spark-plug (to make it suitable for both cold-flow and combustion process simulations).

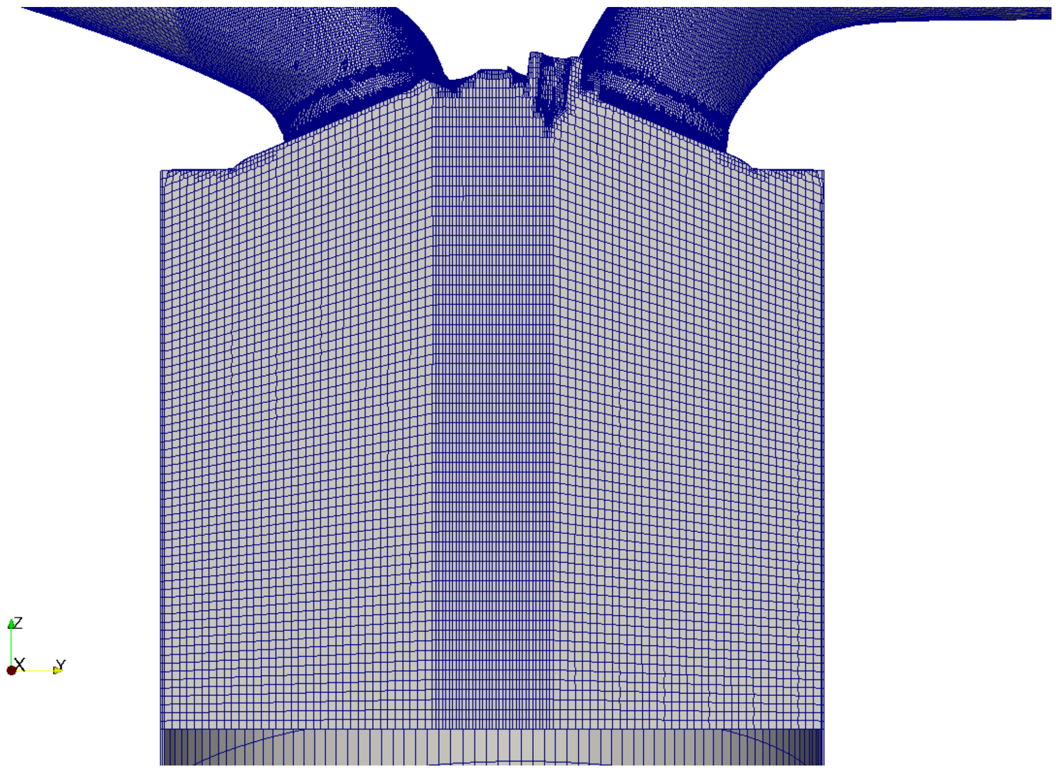


Figure 4.11: In-cylinder mesh.

Finally figure (4.12) shows the internal mesh in correspondence of the valve plane. This specific mesh is at CA 450, mid-intake phase.

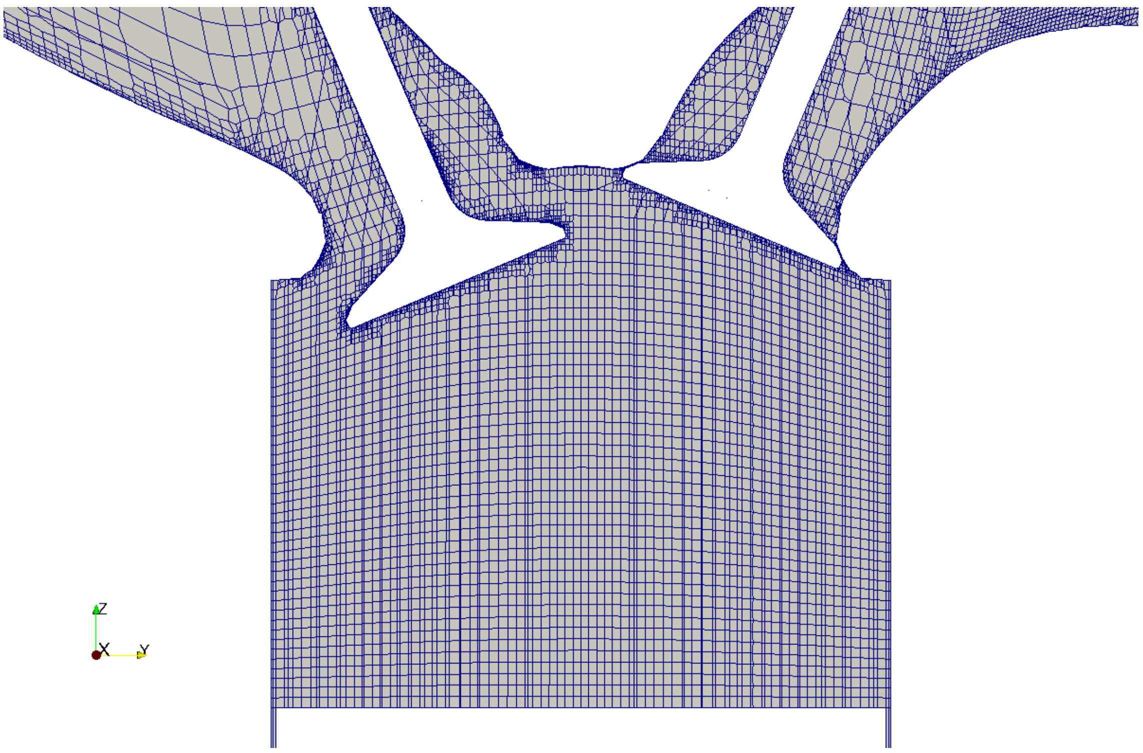


Figure 4.12: Mesh at CA 450, mid-intake phase - valve plane slice.

Chapter 5

Steady-state simulation

In this chapter the performed steady-state simulations are analysed. The first part of the chapter involves the standard engine configuration. Using the available magnetic resonance velocimetry (MRV) measurements as validation, two types of sensitivity analysis have been performed: turbulence model and numerical schemes. In the second part of the chapter the same type of steady-state simulation is carried out on the current engine configuration, to compare the Tumble motion generation in the two engine configurations.

5.1 Experimental validation on the standard engine configuration

This section involves the standard engine configuration, shown in figure (5.1).

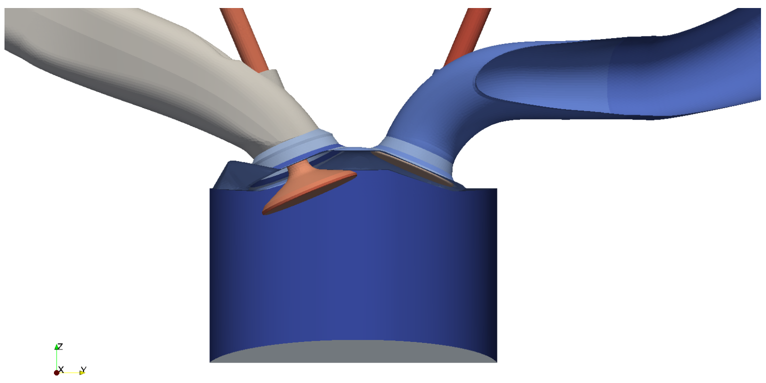


Figure 5.1: Darmstadt engine, standard configuration at CA 450 - cut on the symmetry plane.

The validation has been performed in steady-state condition at CA 450, when the intake valve is at its maximum lift (mid-intake). This choice derives from the availability of magnetic resonance velocimetry (MRV) measurements performed at these conditions on the standard configuration [11], shown in figure (5.2).

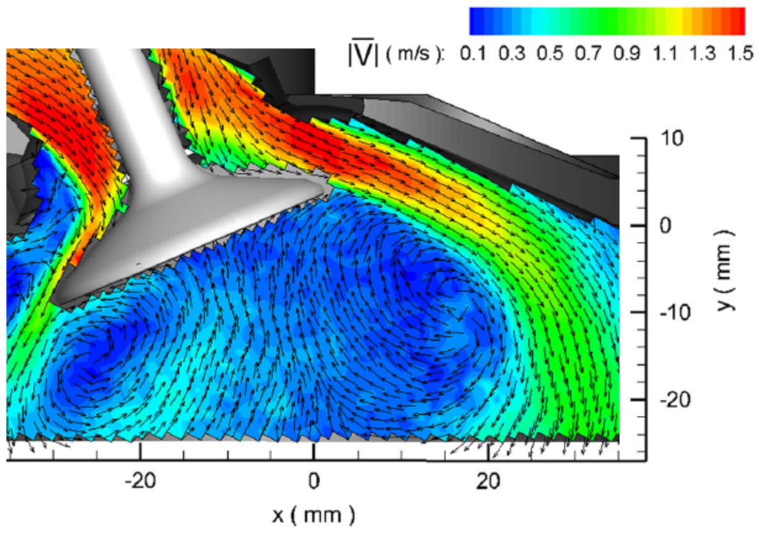


Figure 5.2: MRV measurements performed on the standard engine configuration - intake flow in the vicinity of the intake valve [11].

The flow parameters during the MRV measurements are reported in table (5.1).

Fluid	De-ionized water
Temperature	50 °C
Volumetric flow rate	66L/min

Table 5.1: Flow parameters during the MRV measurements.

It has to be noticed that the working fluid used in the experiments is de-ionized water. In order to perform a simulation consistent with the experimental conditions, the solver used for the simulation is the steady-state incompressible solver **simpleFoam**, available in the official OpenFOAM distribution. This solver employs the SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) algorithm, whose iterative procedure can be summed up as follows:

- set the boundary conditions

- solve the discretized momentum equation to compute the intermediate velocity field
- compute the mass fluxes at the cells faces
- solve the pressure equation and apply under-relaxation
- correct the mass fluxes at the cells faces
- correct the velocities on the basis of the new pressure field
- update the boundary conditions
- repeat till convergence

Unlike the PIMPLE algorithm employed for the full-cycle simulation (explained in section 3.3), the SIMPLE algorithm is not based on a temporal loop and it does not solve density and energy equations.

The mesh has been generated using the same settings employed for the current configuration in the full-cycle set of meshes. A further *subSet* has been applied in order to remove the exhaust ducts from the computational domain, as the exhaust valves are closed and the relative ducts are thus not involved in the simulation. Figures (5.3) and (5.4) show the mesh of the standard configuration.

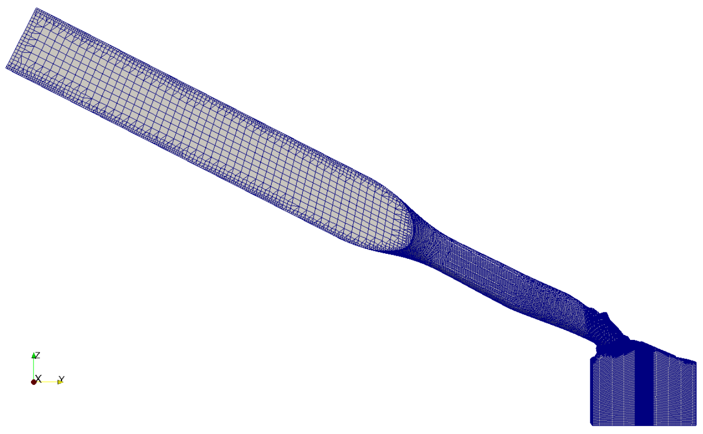


Figure 5.3: Mesh of the standard engine configuration.

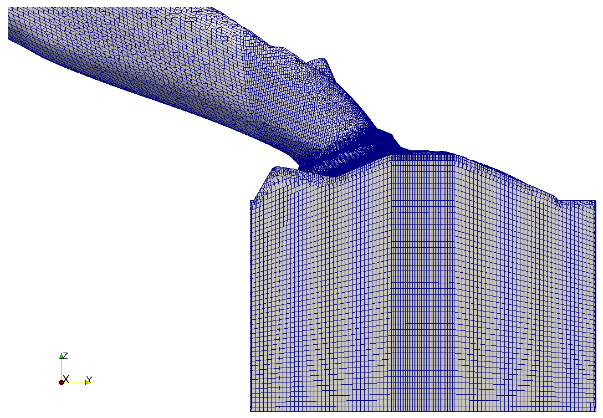


Figure 5.4: Mesh of the standard engine configuration - zoom on the in-cylinder cells.

A sensitivity analysis of turbulence models has been carried out: maintaining the same numerical setting and boundary conditions, two turbulence models have been tested, k-ε and k-ω SST.

Tables (5.2) and (5.3) report the numerical schemes listed in the *fvSchemes* dictionary.

div(phi,U)	bounded Gauss upwind
div(phi,k)	bounded Gauss upwind
div(phi,epsilon)	bounded Gauss upwind
div(phi,omega)	bounded Gauss upwind
div((nuEff*dev2(T(grad(U)))))	Gauss linear

Table 5.2: Applied numerical schemes for the divergence terms.

ddtSchemes	steadyState
gradSchemes	default: Gauss linear grad(U): cellLimited Gauss linear 1.0
laplacianSchemes	Gauss linear limited 0.5
interpolationSchemes	linear
snGradSchemes	limited 0.5

Table 5.3: Applied numerical schemes for the other terms.

The equation solvers and tolerance contained in the *fvSolution* dictionary are listed in tables (5.4) and (5.5).

solver	GAMG
tolerance	1e-7
relTol	0.01
smoother	GaussSeidel
nPreSweeps	0
nPostSweeps	2
cacheAgglomeration	on
agglomerator	faceAreaPair
nCellsInCoarsestLevel	10
mergeLevels	1

Table 5.4: Solver and tolerance for pressure equation.

solver	PBiCG
preconditioner	DILU
tolerance	1e-6
relTol	0.001
nSweeps	2

Table 5.5: Solver and tolerance for U\|k\|epsilon\|omega equations.

For what concerns the boundary conditions, it has to be remarked that the *piston* patch is defined as outlet patch in the steady simulations. Table (5.6) reports the boundary conditions assigned to the *inlet* and *piston* patches and table (5.7) lists the boundary condition assigned to the other patches.

field	inlet	piston
nut	calculated	calculated
k	turbulentIntensityKineticEnergyInlet	inletOutlet
epsilon	turbulentMixingLengthDissipationRateInlet	inletOutlet
omega	turbulentMixingLengthFrequencyInlet	inletOutlet
p	zeroGradient	fixedValue
U	flowRateInletVelocity	pressureInletOutletVelocity

Table 5.6: Boundary conditions assigned to the inlet and piston patches.

field	other patches
nut	nutkWallFunction
k	kqRWallFunction
epsilon	epsilonWallFunction
omega	omegaWallFunction
p	zeroGradient
U	fixedValue

Table 5.7: Boundary conditions assigned to the other patches.

The assigned inlet flow rate is volumetric and equal to 5.5e-4 $[m^3/s]$ (that is half of 66L/min since the simulation is performed on half the engine).

In the following, the results achieved with the two turbulence models are presented and compared.

5.1.1 k-ε turbulence model

The first turbulence model tested is the k-ε model. Figure (5.5) shows the velocity flow field entering the combustion chamber in the standard configuration.

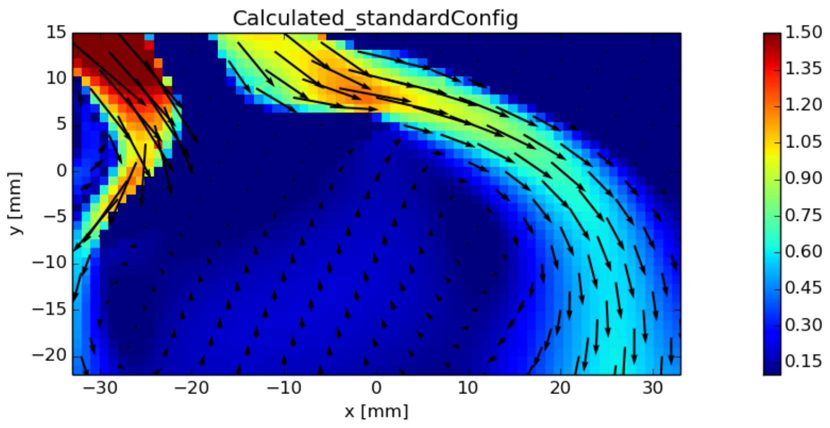


Figure 5.5: Velocity flow field entering the combustion chamber - k-ε model.

Comparing figure (5.5)[1] with the experimental results shown in figure (5.2), it is possible to verify the reliability of this turbulence model. Both the values and the evolution of the velocity field are consistent with the MRV measurement, even if in the numerical result the flow undergoes a higher deceleration once in the combustion chamber, as shown in figure (5.6).

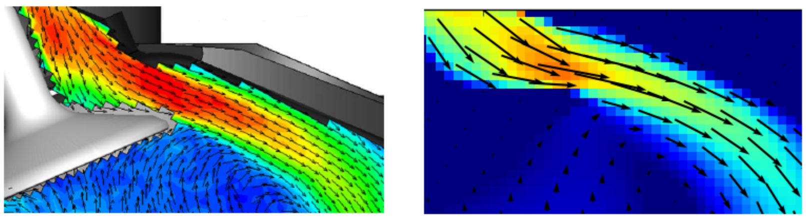


Figure 5.6: Zoom on the difference in the flow field once in the combustion chamber: the flow undergoes a higher deceleration in the numerical result (on the right).

This effect however is probably caused by the chosen numerical schemes, which aim at a high stability more than a high accuracy. The achieved result is in line with the literature: the k-ε model is widely used in internal combustion engine simulations [12].

[1]the image resolution depends on the resolution of the sampling (that is 1 mm), not on the grid resolution.

5.1.2 k-ω SST turbulence model

The second turbulence model tested is the k-ω SST model. Figure (5.7) shows the velocity flow field entering the combustion chamber in the standard configuration.

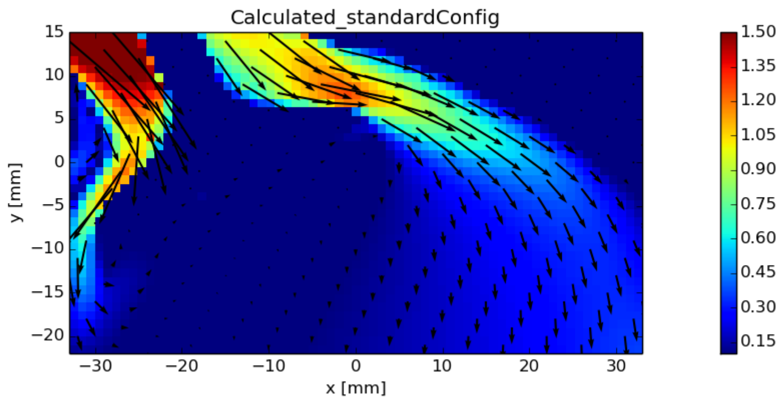


Figure 5.7: Velocity flow field entering the combustion chamber - k-ω SST model.

Even if this model should combine the strengths of the k-ε (away from wall) and k-ω (near walls) models, its prediction of the entering flow field is not satisfactory. As shown in figure (5.8), in this case the flow undergoes a strong deceleration once in the combustion chamber: a Tumble motion generation would not be possible with the calculated velocity field.

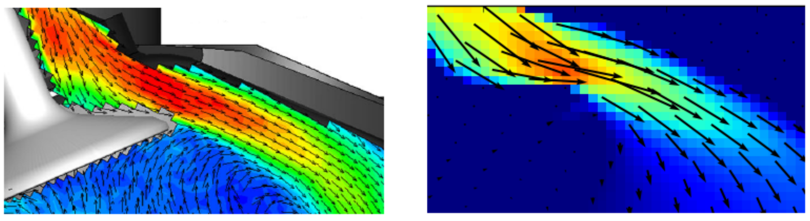


Figure 5.8: Zoom on the difference in the flow field once in the combustion chamber: the flow undergoes a strong deceleration in the numerical result (on the right).

Comparing the achieved results with the experimental measurements, k-ε model proves to be better than the k-ω SST in the prediction of the incoming flow, as already stated by [1] in a similar analysis.

The achieved values of the nondimensional distance from wall $y^+ = \frac{y}{\delta_\nu}$ (where δ_ν is the viscous length-scale) however are lower than the suggested threshold for the k-ε model ($y^+ \geq 11.225$ [2]). The maximum values on the involved patches are around 5 (value still accepted), but the mean values are around 0.5. For this reason a further simulation has been performed, applying to the k-ε model the *scalableWallFunction*. The purpose of scalable wall functions is to force the usage of the log-law in conjunction with the standard wall functions approach. This is achieved by introducing a limiter in the y^+ calculations:

$$y^{+^{\sim}} = MAX(y^+, y^+_{lim}) \tag{5.1.1}$$

where $y^+_{lim} = 11.225$, value of the transition to the log-law layer [2]. This wall function virtually displaces the mesh to a $y^+ \sim 11.225$, irrespective of the level of refinement, thereby avoiding the erroneous modelling of the laminar sub-layer and buffer region. The result achieved using the scalable wall functions is shown in figure (5.9).

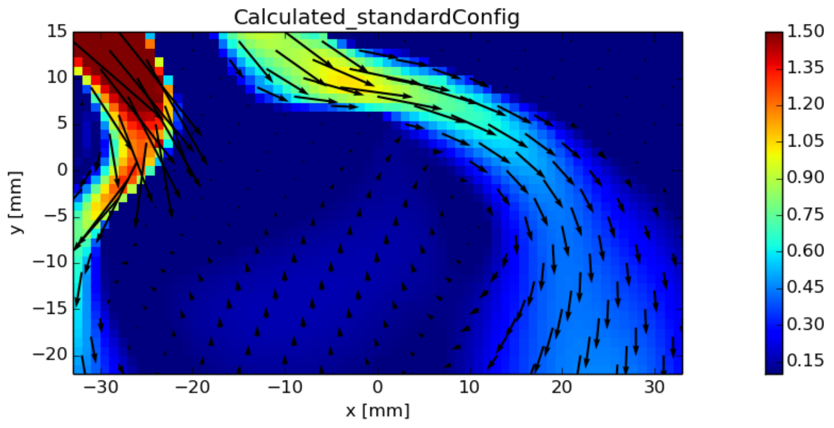


Figure 5.9: Velocity flow field entering the combustion chamber - k-ε model with scalable wall functions.

It can be noticed that using the scalable wall functions the flow entering the combustion chamber undergoes a deceleration higher than the one achieved in the standard k-ε model and lower than the one resulting from the k-ω SST. The best agreement with the experimental data (figure 5.2) is achieved by the standard k-ε model (figure 5.5). The velocity magnitude justifies the low y^+ evaluated with the standard k-ε model and in this analysis the shear stress at the wall does not play an important role.

5.1.3 Improved numerical schemes

Once established that the k-ε model is the most suitable for this type of simulation, a further sensitivity analysis has been carried out. The aim was to verify the effect of the numerical schemes on the observed deceleration of the flow field inside the combustion chamber.

For this reason a further simulation has been performed using the k-ε turbulence model and the same settings employed for the previous simulations, except for the numerical schemes for the divergence terms. The modified schemes tested in this simulation are listed in table (5.8).

div(phi,U)	Gauss linearUpwindV grad(U)
div(phi,k)	Gauss limitedLinear 1.0
div(phi,epsilon)	Gauss limitedLinear 1.0
div(phi,omega)	Gauss limitedLinear 1.0
div((nuEff*dev2(T(grad(U)))))	Gauss linear

Table 5.8: Modified numerical schemes for the divergence terms.

As explained in section (1.5.1), the linear interpolation has an accuracy of the second order, while the upwind scheme used in the previous settings has an accuracy of the first order (but a higher stability). The linearUpwind interpolation is a combination of these schemes having higher accuracy still preserving stability.

Figure (5.10) shows the velocity flow field entering the combustion chamber achieved with the improved numerical schemes.

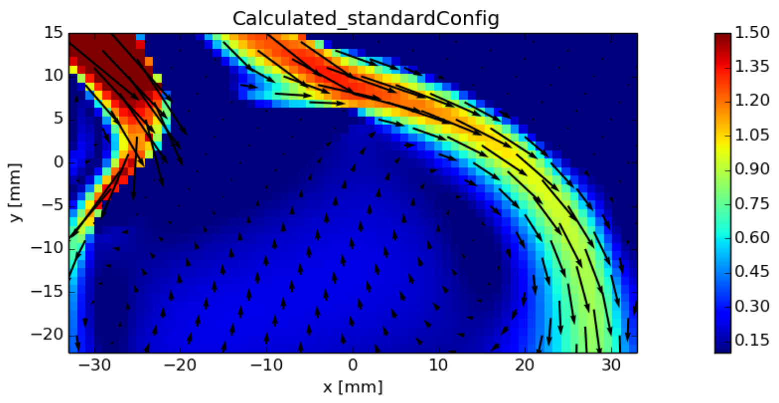


Figure 5.10: Velocity flow field entering the combustion chamber - k-ε model, improved numerical schemes.

As highlighted in figure (5.11), in this case the velocity value is correctly predicted not only on the valve curtain but also inside the cylinder: the incoming flow does not undergo the previous observed deceleration (white oval). Moreover the artificial diffusivity is reduced, leading to a lower spread of the flow over the grid in correspondence of the cylinder walls on the exhaust side (white rectangle).

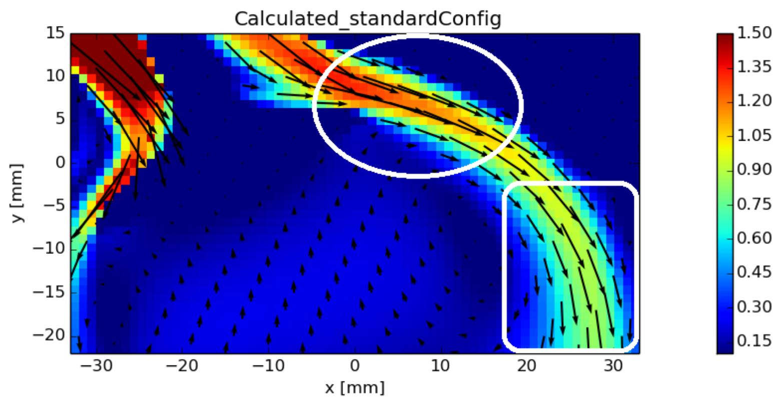


Figure 5.11: Velocity flow field entering the combustion chamber - improvements highlighted.

Considering the satisfactory results achieved in the steady-state simulation, the same type of settings (k-ε model, improved numerical schemes) have been employed in the full-cycle simulation.

5.2 Engine configurations comparison

In this section the two engine configurations are compared: a steady-state simulation with the optimal setting (section 5.1.3) has been performed on the current engine configuration to allow a consistent comparison. As it is possible to observe from figure (5.12), the two configurations differ mainly in the orientation of the final part of the intake duct. In addition the intake valve diameters are slightly reduced (from 33 mm to 31 mm) in the current configuration and even the cylinder-head has been changed on the intake side. Differences in the in-coming velocity field are thus expected.

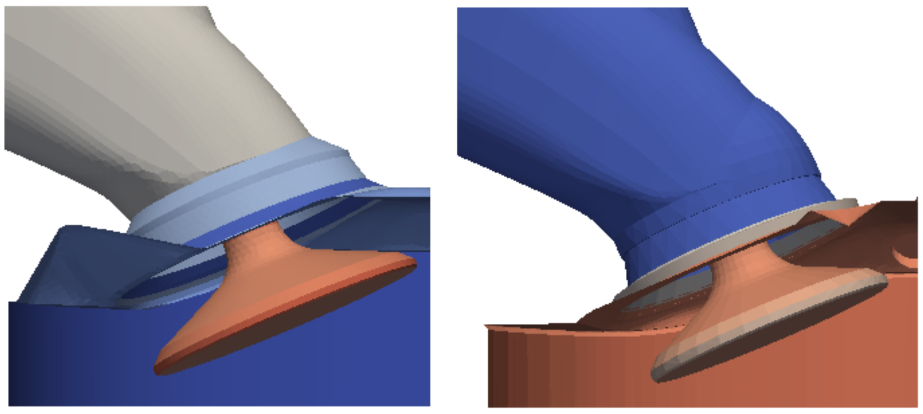
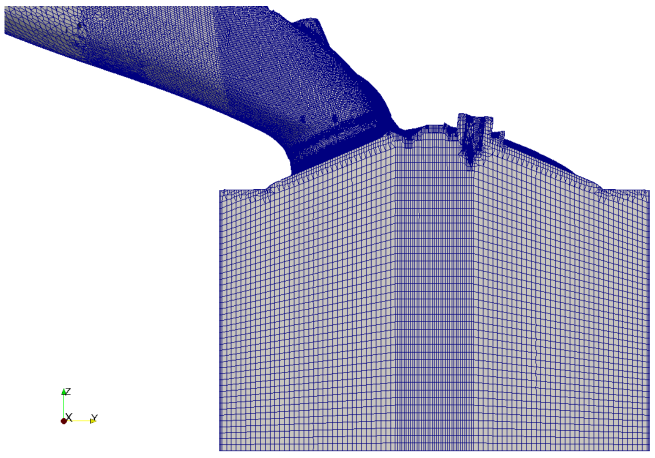


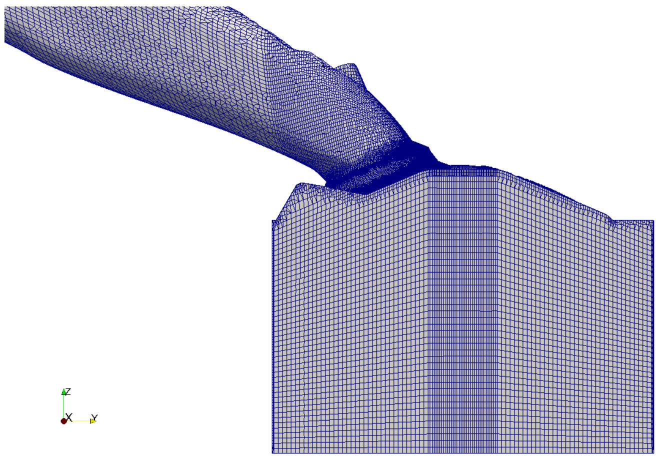
Figure 5.12: Zoom on the final part of the intake duct of the standard (left) and current (right) configuration.

The mesh of the current configuration is taken from the full-cycle set of meshes (at CA 450), generated as explained in chapter (4). As specified in the first part of this chapter, the mesh of the standard configuration has been generated with the same settings (same background mesh and same settings for the body-fitted mesh). The in-cylinder mesh of the two configurations is shown in figure (5.13)

It can be noticed that the grids are perfectly comparable: the results will not be affected by the mesh, the expected differences in the flow field will be a consequence of the geometry only.



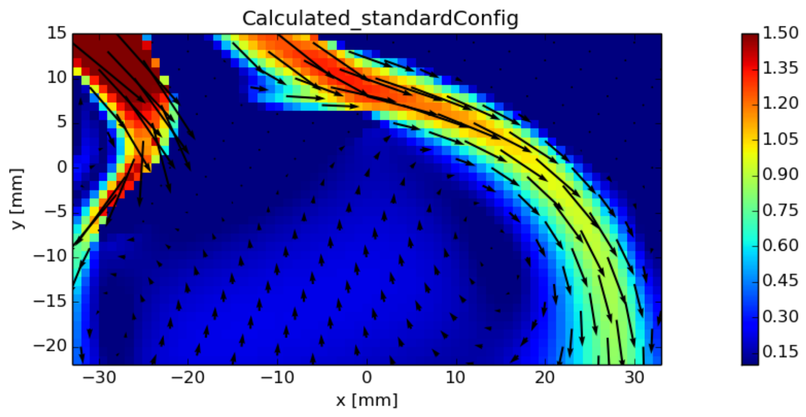
(a) Mesh of the current engine configuration.



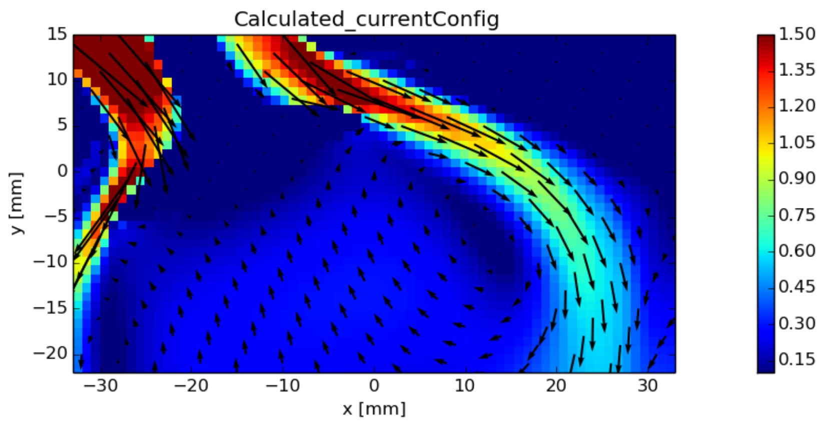
(b) Mesh of the standard engine configuration.

Figure 5.13: Comparison of the mesh of the two configurations.

In figure (5.14) are shown the velocity flow fields entering the combustion chamber in the standard and in the current configurations respectively.



(a) Velocity flow fields entering the combustion chamber - standard engine configuration.



(b) Velocity flow fields entering the combustion chamber - current engine configuration.

Figure 5.14: Comparison of the velocity flow fields entering the combustion chamber mesh in the two configurations.

Analysing this figure it is possible to notice two main differences in the velocity flow field, highlighted in figure (5.15).

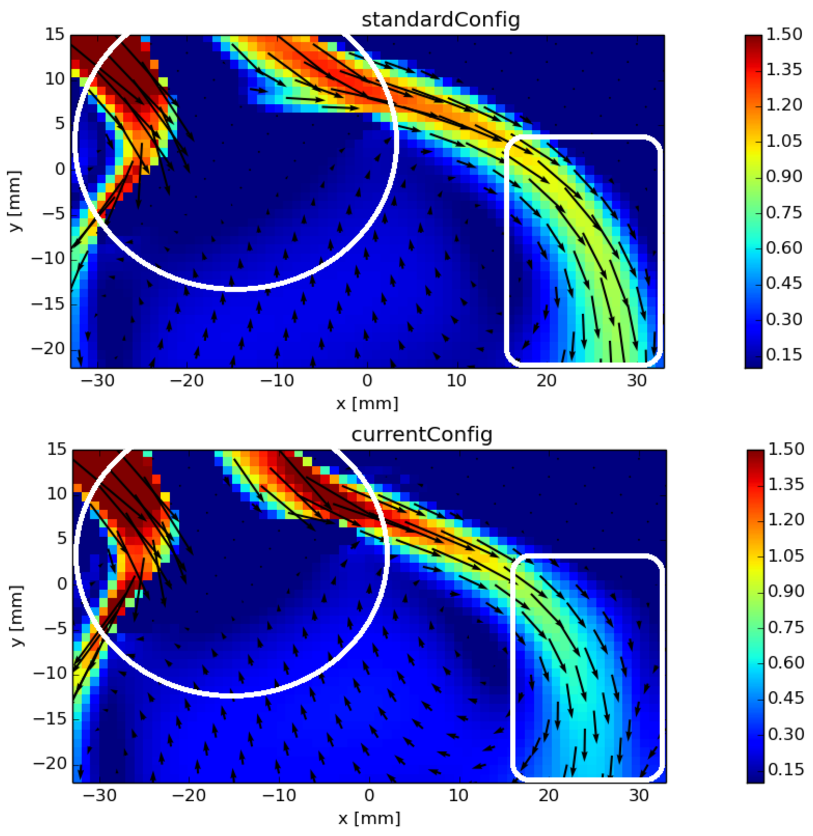


Figure 5.15: Comparison of the velocity flow fields entering the combustion chamber mesh in the two configurations - differences highlighted.

The first difference involves the velocities close to the intake valve, highlighted by the white circle in figure (5.15) and zoomed in in figure (5.16).

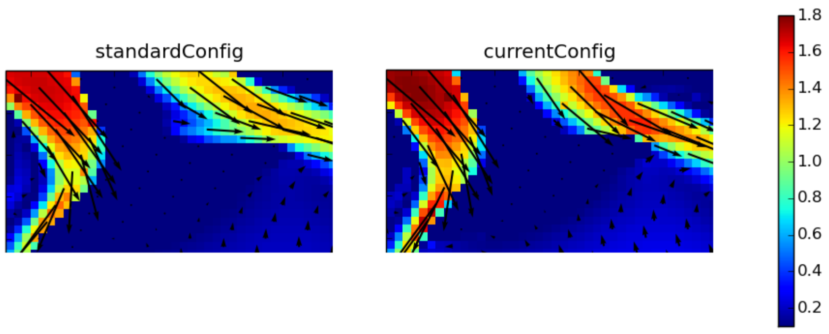


Figure 5.16: Velocity difference close to the intake valve - zoom.

In figure (5.16) the range of velocity values have been re-scaled to better appreciate the difference in the two configuration. In the current configuration the velocity reaches slightly higher values thanks to the smaller valve diameter.

The second difference involves the flow field evolution inside the cylinder, highlighted by the white rectangle in figure (5.15) and zoomed in in figure (5.16).

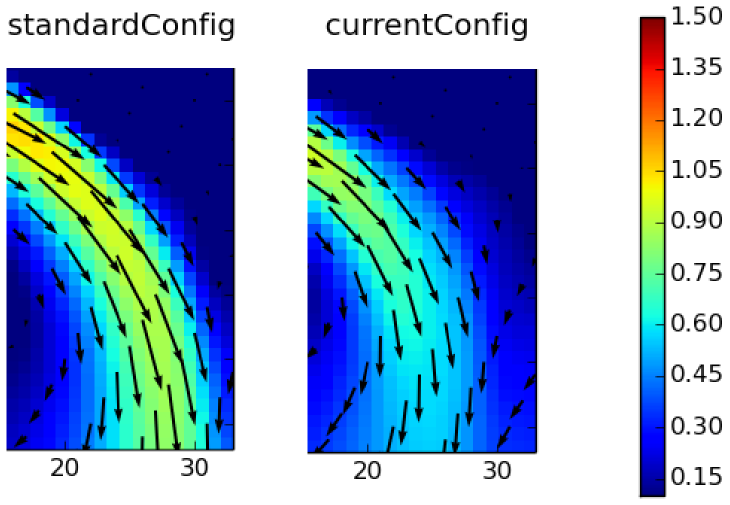


Figure 5.17: Velocity difference close to the cylinder walls on the exhaust side - zoom.

In the current configuration the flow is not driven towards the cylinder walls on the exhaust side as much as in the standard configuration. This difference derives from the orientation of the final part of the intake duct: in the current configuration the duct ends more uprightly, leading to a direct vortex less oriented towards the cylinder walls.

Finally it can be observe from figure (5.15) that in the current configuration the velocity of the flow pouring out of the left side of the intake valve is higher. This flow will generate a counter-vortex more intense that will soften the direct Tumble motion.

Considering these differences it can be deduced that the standard configuration generates a higher Tumble motion than the current one.

In figure (5.18), taken from [8], a quantitative study of the effects of the intake port orientation on the Tumble generation is shown.

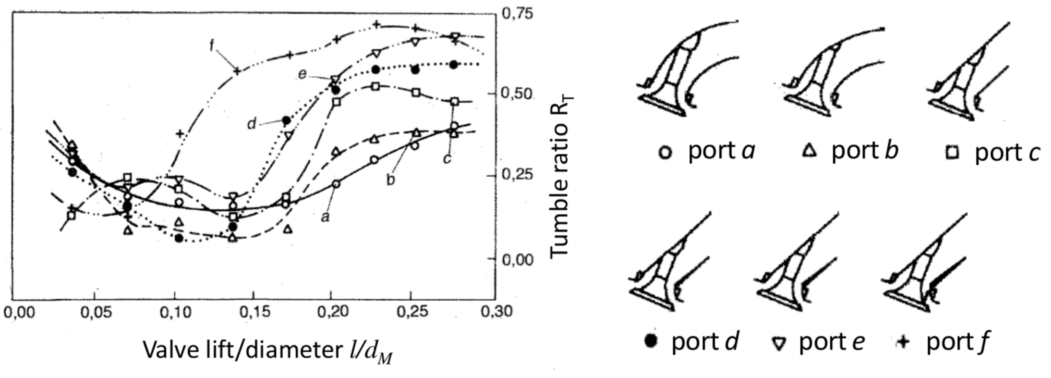


Figure 5.18: Tumble ratio for different port configurations, figure taken from [8].

As shown in figure (5.18), the port configurations generating higher Tumble motion (higher Tumble ratio) are the ports *d*, *e* and *f*, which are similar to the port configuration of the standard engine (see figure (5.19)). On the contrary, ports *a* and *b*, which are comparable to the port configuration of the current engine, generate a lower Tumble motion (lower Tumble ratio).

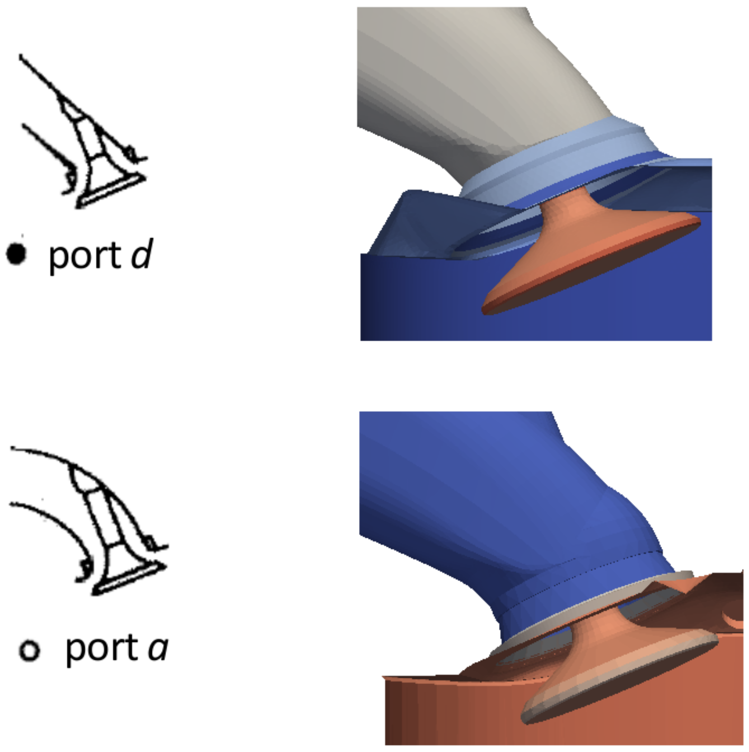


Figure 5.19: Comparison between the engine configurations and the port types analysed in figure (5.18).

Chapter 6

Full-cycle simulation

In this chapter the full-cycle simulation is analysed. This simulation is run automatically by the application **runMultiCycleCase**, which is perfectly coupled with the **engineDynamicSetUp** application responsible for the full-cycle mesh generation.

In the first part the simulation settings are reported. In the second part the evolution of the Tumble motion is analysed and compared to the one in the standard configuration. Finally the cylinder pressure trend is taken into account.

6.1 Simulation settings

runMultiCycleCase application runs the gas exchange simulation with the settings predefined in the *engineControlDict* dictionary (initial case, start- and end-times, time step and write intervals). Similarly to **engineDynamicSetup**, **runMultiCycleCase** works in loop and in every mesh-case the steps are:

- execution of the cold flow simulation: **simpleColdSpeciesEngineDYMFoam** solver
- reconstruction of the domain in parallel configuration: **reconstructParEngine** utility
- case to case interpolation of the field conditions: **engineMapFields** utility
- decomposition of the domain in parallel configuration: **decomposeParEngine** utility

As explained in section (2.2), **simpleColdSpeciesEngineDYMFoam** is a standard solver for cold-flow simulations, able to handle the mesh motion.

In each case the numerical settings, the physical properties and the boundary and initial conditions are defined. These settings have to be specified only once in the **init** folder before running the **engineDynamicSetup** application: this folder is then automatically copied in each case folder (see sections 3.2.2 and 3.3).

The numerical settings are defined in *fvSchemes* and *fvSolution* dictionaries stored in the **system** folder (present in each case). The numerical schemes listed in *fvSchemes* are similar to the ones employed in the steady-state simulation (section 5.1.3). In this case the solver is transient and compressible, therefore more terms are required. Tables (6.1) and (6.2) list the numerical schemes for the divergence terms and for the other terms respectively.

div(phi,U)	Gauss linearUpwindV grad(U)
div(phiU,p)	Gauss limitedLinear 1.0
div(phi,Yi_h)	Gauss limitedLinear 1.0
div(phid,p)	Gauss limitedLinear 1.0
div(phi,k)	Gauss upwind
div(phi,epsilon)	Gauss upwind
div((muEff*dev2(T(grad(U)))))	Gauss linear
div(U)	Gauss linear
div(meshPhi,p)	Gauss linear
div(mesh,p)	Gauss linear

Table 6.1: Numerical schemes for the divergence terms.

ddtSchemes	Euler
gradSchemes	cellLimited Gauss linear 1.0
laplacianSchemes	default: Gauss linear limited 1.0 laplacian(DkEff,k): Gauss linear limited 0.5 laplacian(DepsilonEff,epsilon): Gauss linear limited 0.5
interpolationSchemes	linear
snGradSchemes	limited 1.0

Table 6.2: Numerical schemes for the other terms.

The main equation solvers (listed in *fvSolution*) applied are:

- GAMG for pressure and cellMotionU
- PCG for density
- PBiCG for velocity, enthalpy, turbulence quantity

For what concerns the boundary and initial conditions, particular attention has to be paid to pressure and velocity fields. A time-varying pressure is assigned to the *inlet* and *outlet* patches, as shown in table (6.3).

patch	type	timeDataFileName
inlet	engineTimeVaryingTotalPressure	$FOAM_CASE/../dataTime/p_man2.t
outlet	engineTimeVaryingTotalPressure	$FOAM_CASE/../dataTime/p_exh1.t

Table 6.3: Pressure condition assigned to the inlet and outlet patches.

The velocity dictionary must be coherent with the moving boundaries, as shown in table (6.4).

patch	type
inlet|outlet	pressureInletOutletVelocity
piston	movingWallVelocity
valve-Side|Bottom|Top	movingWallVelocity
others	fixedValue

Table 6.4: Velocity boundary conditions.

where the valve statement includes both the intake and exhaust valve.

6.2 Results

In this section the achieved results are analysed. In particular great attention has been paid to the Tumble motion generation and evolution. A comparison with the experimental data have been performed at CA 450, where the experimental data provided by TU Darmstadt are available. Then, the current and standard configurations have been compared in terms of Tumble generation at different CA, using the results achieved by ICE Group on the standard configuration during a previous work. Finally some considerations on the in-cylinder pressure trend are reported.

6.2.1 Tumble motion

In this section three important phases of the Tumble motion generation and evolution are analysed. As already explained in section (1.2), the Tumble motion is generated during the intake phase but maintained and increased by the following compression stroke. For this reason the analysis have been performed at CA 450 (mid-intake), 540 (intake phase, piston at BDC) and 630 (mid-compression).

For each CA, two types of visual comparisons are carried out on the symmetry plane:

- overall velocity flow field inside the combustion chamber. In this case the analysis is not performed on the entire combustion chamber, but where experimental data are available (optical window described in section 3.1)
- x and y velocity components along four different measurement lines located at different distances from the cylinder head, shown in figure (6.1). In this case the analysis covers the entire bore (86 mm) if the comparison involves only computed results

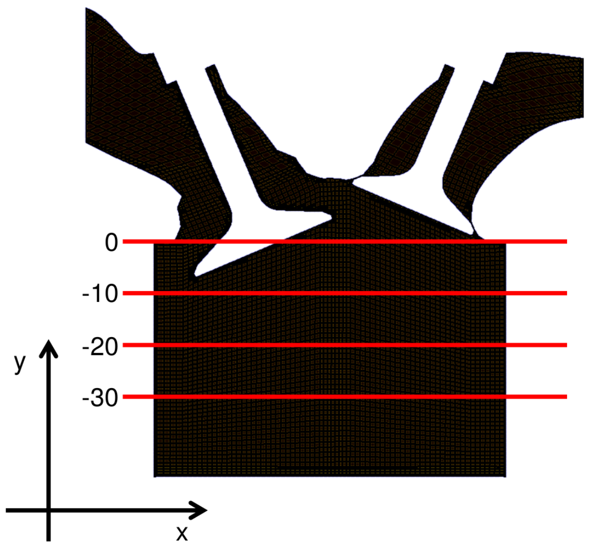


Figure 6.1: Measurement lines located at different distances from the cylinder head.

CA 450, mid-intake

The comparison between experimental and computed velocity flow field at CA 450 is shown in figure (6.2).

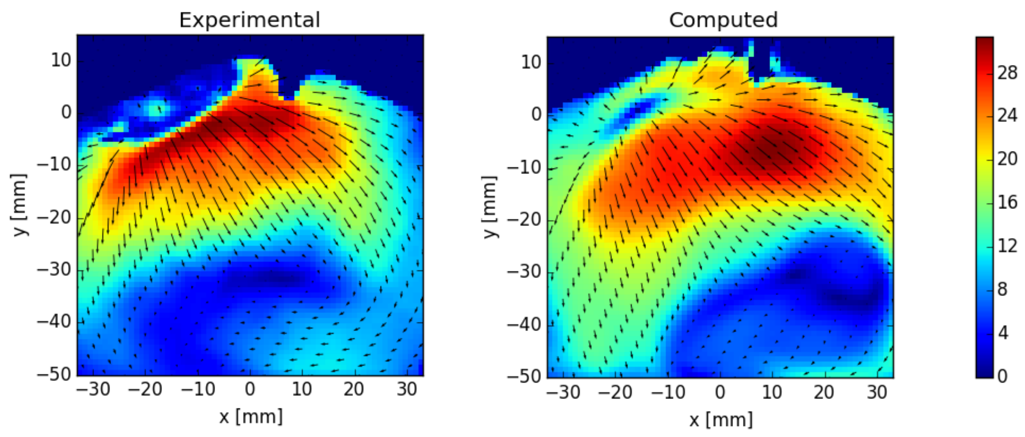


Figure 6.2: Comparison between experimental (left) and computed (right) velocity field on the symmetry plane at CA 450.

The incoming flow interacts with the cylinder liner moving towards the piston, generating the Tumble vortex. It can be observed that the velocity magnitude is well predicted, but the peak is more shifted towards the exhaust side. On the left side of the cylinder ($-30 < x < -10$ mm), the counter-vortex is overestimated: in the computed result the region where the y-component of the velocity in downward direction exceeds the absolute value of $U_y = 10$ is wider, as it possible to observe in the following graphics. On the other hand, on the bottom-right side of the cylinder ($x > 0$, $-50 < y < -30$ mm) the direct vortex begins to rise: in the experimental field it can be observed that the flow begins to rotate, moving horizontally towards left instead of downwards. This rotation, which takes place for $-50 < y < -30$ and $0 < x < 30$ mm in the experimental field, is reduced in the computed field, where an horizontal velocity begins to appear only at $y \sim -50$ mm.

In order to obtain a quantitative analysis of the phenomena qualitatively observed in figure (6.2), the velocity components extracted along horizontal lines at different distances from the cylinder head are shown in figures (6.3) and (6.4).

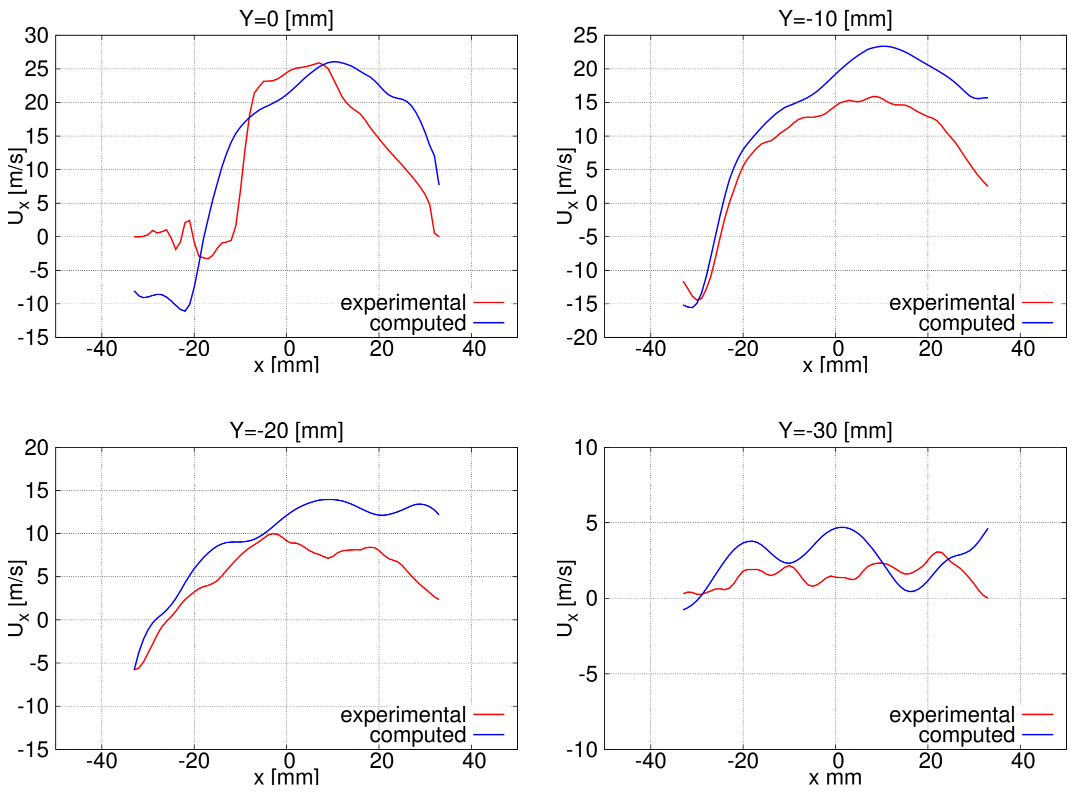


Figure 6.3: Velocity x-component profiles at CA 450, extracted along horizontal lines at different distances from the cylinder head.

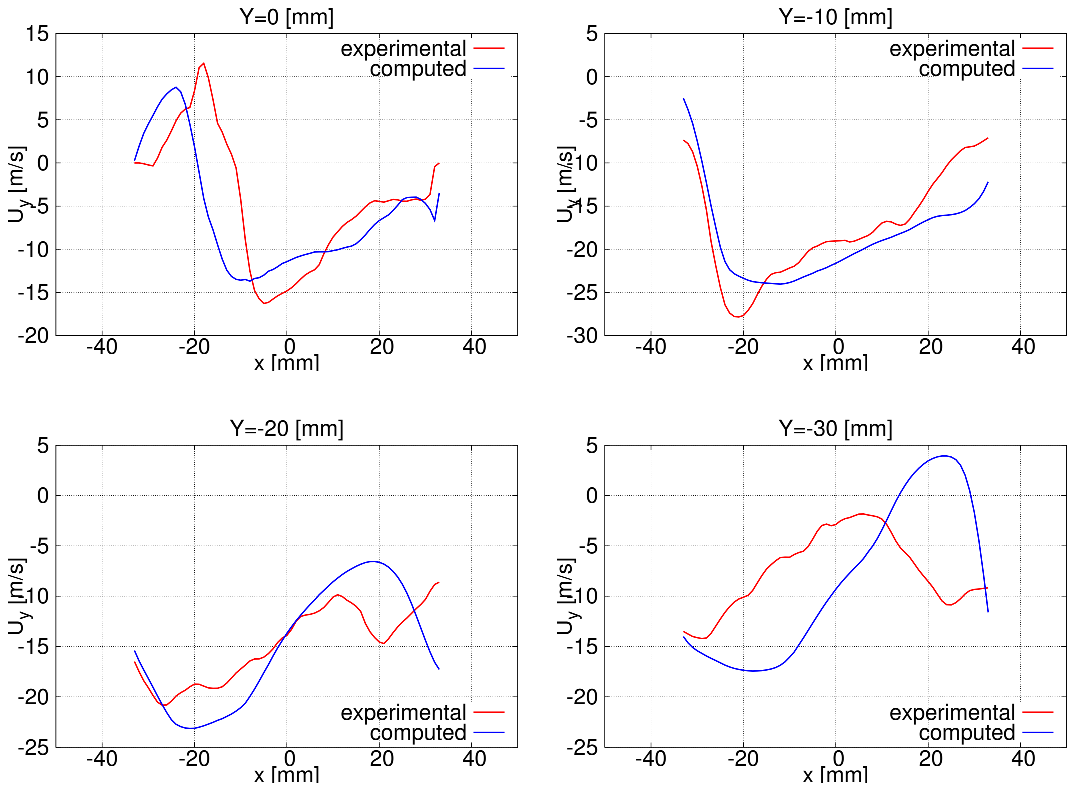


Figure 6.4: Velocity y-component profiles at CA 450, extracted along horizontal lines at different distances from the cylinder head.

It is possible to observe that there is a quite good agreement between the experimental and computed velocity components.

At $Y = 0$ and $Y = -10$ mm the trend of both the x- and y-component is in line with the experimental one. The velocity peak more shifted to the right is testified by higher values of the computed curve in the right part of the cylinder ($x > 0$). It has to be noticed that only the x-component of the velocity is responsible for the translation of the velocity peak, while the y-component finds a good quantitative agreement with the corresponding experimental data.

Moving towards the piston the difference increases, still maintaining a sufficient agreement. At $Y = -20$ mm the x-component of the computed velocity shows complete agreement with the experimental one. For $x > 0$ instead the computed field is driven toward the liner more than the experimental one, which begins to be affected by the rotation taking place close to the piston. Similar behaviour is observable at $Y = -30$ mm. The y-component of the velocity exhibits bigger differences, in particular at $Y = -30$ mm: for $-30 < x < -10$ mm the trend of the computed and experimental curves is opposite, testifying the higher intensity of the counter vortex still present at $Y = -30$ mm in the computed results.

A remarkable difference between experimental and computed velocity fields appears at $Y = -40$ mm, as highlighted in figure (6.5).

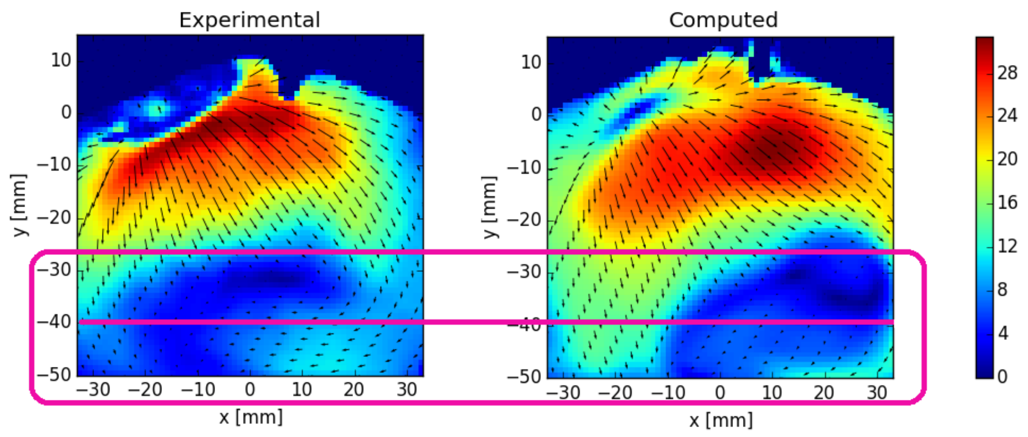


Figure 6.5: Comparison between experimental (left) and computed (right) velocity field on the symmetry plane at CA 450 - zone with biggest differences highlighted.

For this reason the velocity components extracted along $Y = -40$, shown in figure (6.6), are further analysed at this CA.

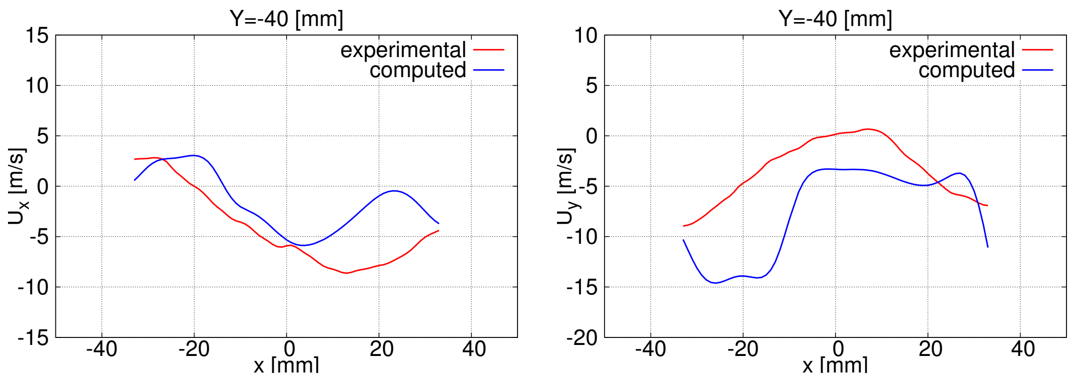


Figure 6.6: Velocity x- and y-components profiles at CA 450, extracted along horizontal line at $Y = -40$ mm.

The x-component of the computed velocity is in good agreement with the experimental one in the intake side of the cylinder ($x < 0$), but it has an opposite trend in the exhaust side: in the experimental field the Tumble vortex begins to rise leading the flow to rotate toward left, while in the computed field this effect is still not present.

The y-component presents differences along the entire bore. For $-40 < x < -15$ mm the trend of computed and experimental curves is opposite: the computed flow moves towards the piston with higher velocity, while the experimental flow begins to rotate toward the center. On the right, the experimental flow is already developed in a vortex slightly moving upwards, leading to a null/positive y-component on the cylinder axis ($x = 0$). The computed flow instead begins to move towards left only at $y = -50$ mm: at this sampling line it is still moving downwards, the y-component is fully negative.

A further comparison has been performed between standard and current configurations. Considering the comparison carried out in steady-state conditions, similar result is expected: a higher Tumble motion generated in the standard engine configuration.

The results on the standard configuration are taken from the previous work done by the ICE Group [5, 16]. The reported results are achieved on a flow-oriented grid with a resolution similar to the one employed in the present work. For the standard configuration, experimental data were available at CA 450, 540 and 630. Figure (6.7) shows the comparison between experimental data and computed results on the standard configuration.

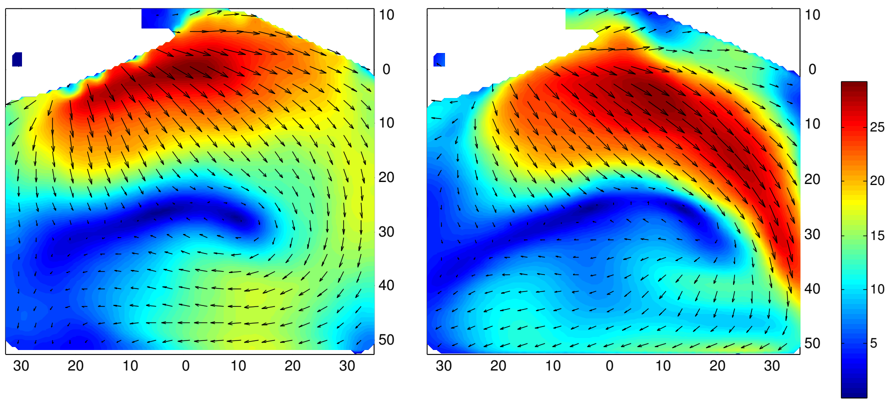


Figure 6.7: Comparison between experimental (left) and computed (right) velocity field on the symmetry plane at CA 450, taken from [16].

Even in this case the computed velocity peak is shifted to the right. The counter-vortex on the left ($-30 < x < -10$ mm) instead is underestimated in the computed field of [16]. Comparing figures (6.7) and (6.2) the difference in the Tumble generation is remarkable, starkly higher in the standard engine configuration, in keeping with the analysis performed in steady-state condition. In figures (6.8) and (6.9) are shown the experimental and computed velocity components extracted along horizontal lines at different distances from the cylinder head in the standard configuration.

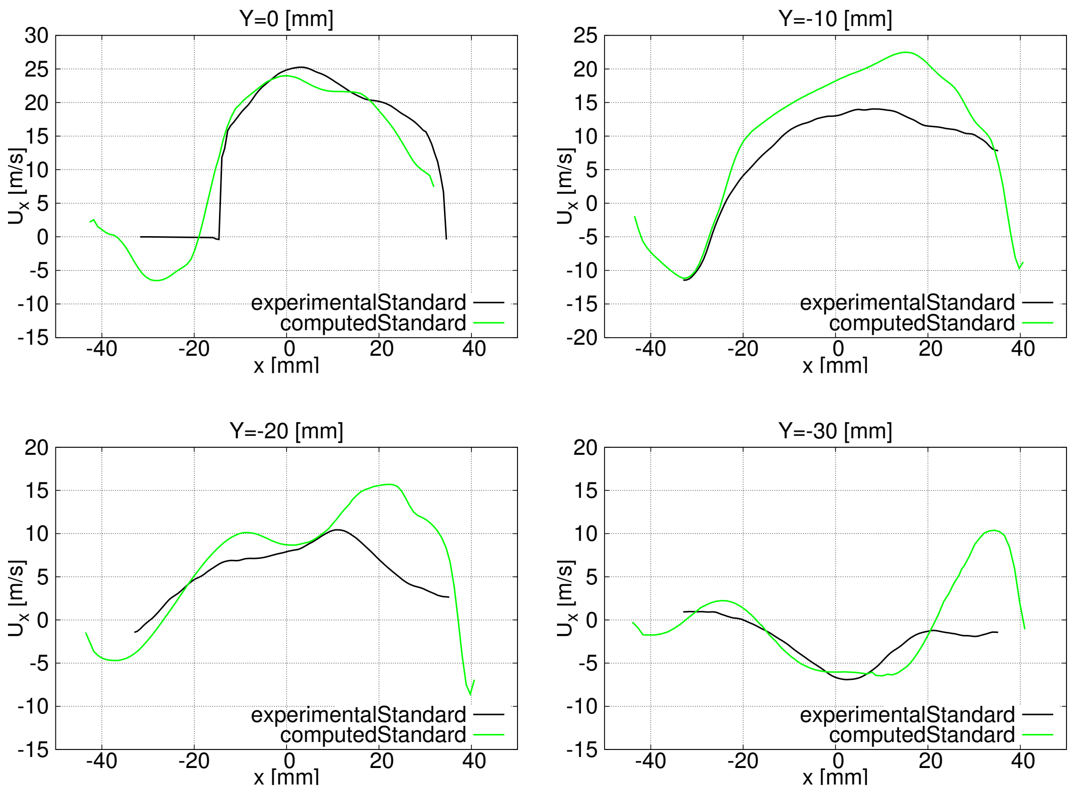


Figure 6.8: Velocity x-component profiles at CA 450 - standard configuration.

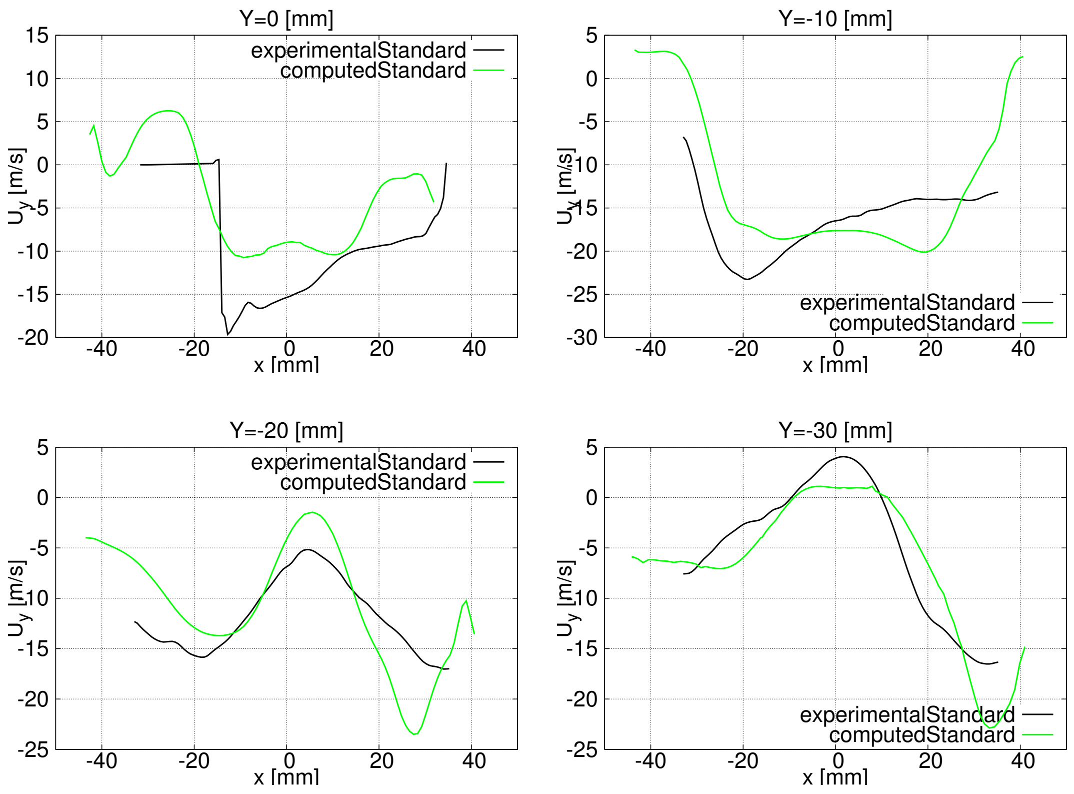


Figure 6.9: Velocity y-component profiles at CA 450 - standard configuration.

First it can be observed that the deviation of the computed curves from the experimental ones in the analysis performed on the standard configuration [16] is comparable with the corresponding deviation in the analysis carried out on the current one. The achieved agreement between measurements and numerical results can thus be regarded as good, in line with [16]. For this reason it can be assumed that in the following comparisons, involving the computed results only, the geometry is mainly responsible for the differences between the curves.

A second remark involves the differences between the two experimental curves, one representing the experimental data of the current configuration and one representing the experimental data of the standard one, compared in figures (6.10) and (6.11).

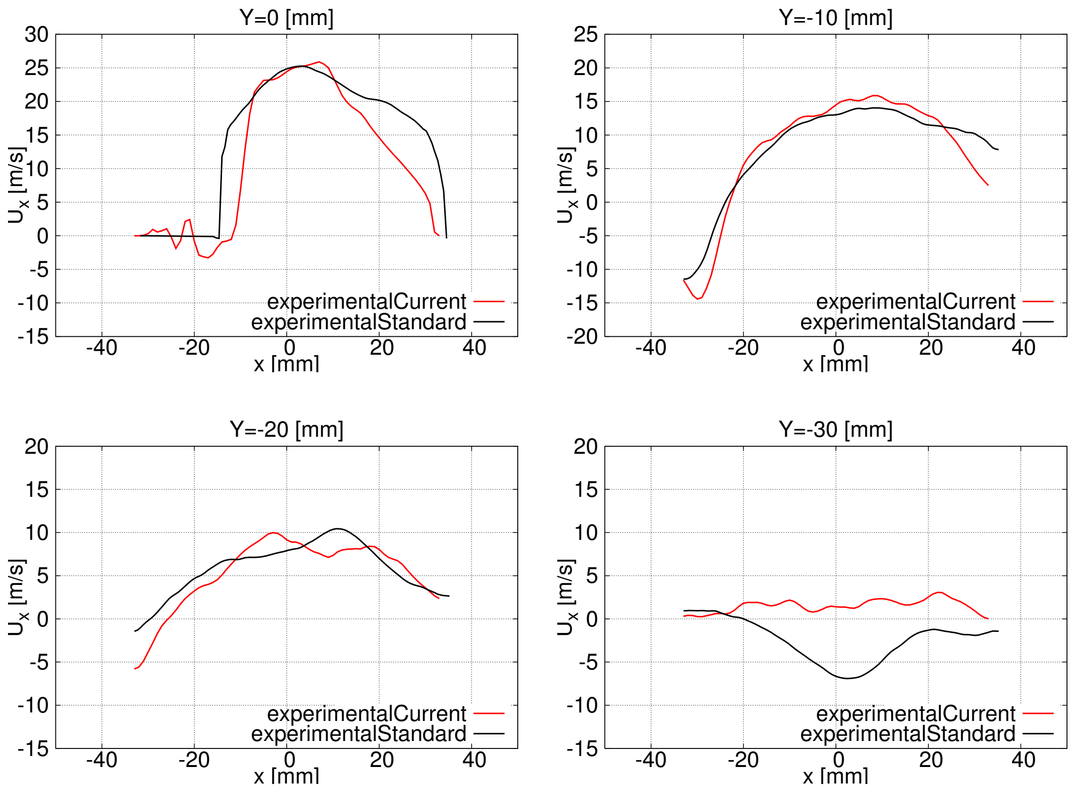


Figure 6.10: Velocity x-component profiles at CA 450 - experimental data of both configurations.

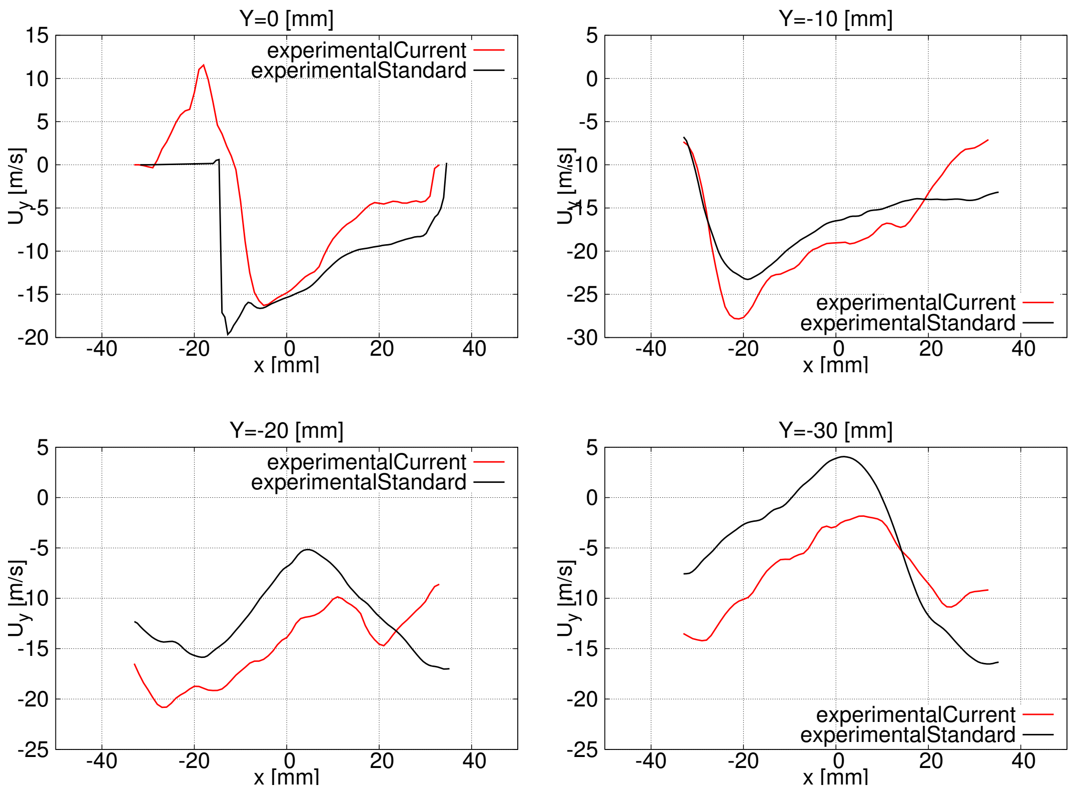
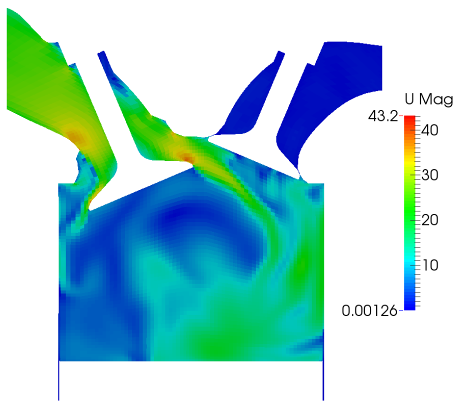


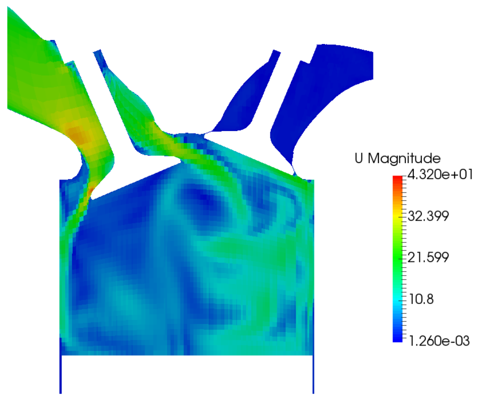
Figure 6.11: Velocity y-component profiles at CA 450 - experimental data of both configurations.

As expected, there is a remarkable deviation of the red curve from the black one, especially at $Y = -30$ mm: at this distance the Tumble vortex begins to rise, thus the different capability to generate the structured motion stands out. The x-component of the velocity in the standard configuration is negative along the entire bore, while in the current configuration it is still positive: in the standard configuration the flow has rotated towards left ($U_x < 0$) generating the Tumble vortex. The trend of the y-component leads to the same conclusion: in the standard configuration the y-component reaches positive values in the central part of the cylinder ($x \sim 0$) because the flow has already undergone the "tumble" and it begins to move upwards, while in the current configuration the y-component is still negative along the entire bore.

Figure (6.12) shows the computed velocity flow field on the valve plane, in the standard (result taken from [16]) and current configurations .



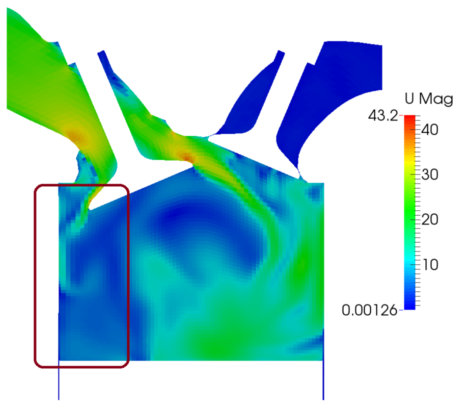
(a) Standard configuration, figure taken from [16].



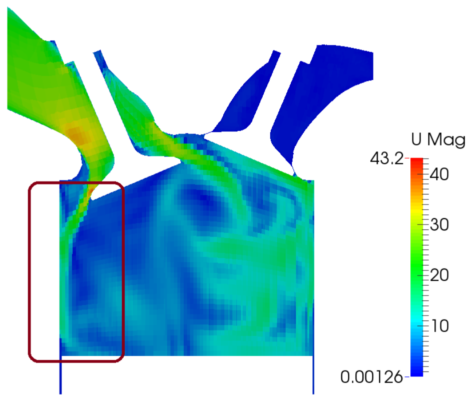
(b) Current configuration.

Figure 6.12: Computed velocity flow field on the valve plane in the standard and current engine configurations.

This figure further confirms what already observed: as highlighted in figure (6.13), the intensity of the counter-vortex is remarkably higher in the current engine configuration and the direct vortex is not as driven toward the cylinder liner on the exhaust side as in the standard configuration.



(a) Standard configuration, figure taken from [16].



(b) Current configuration.

Figure 6.13: Computed velocity flow field on the valve plane in the standard and current engine configurations - counter-vortex highlighted.

CA 540, intake at BDC

At CA 540 the experimental data on the current configuration are not available. A comparison between current and standard configurations is thus performed. Figure (6.14) shows the computed velocity flow field in the current configuration.

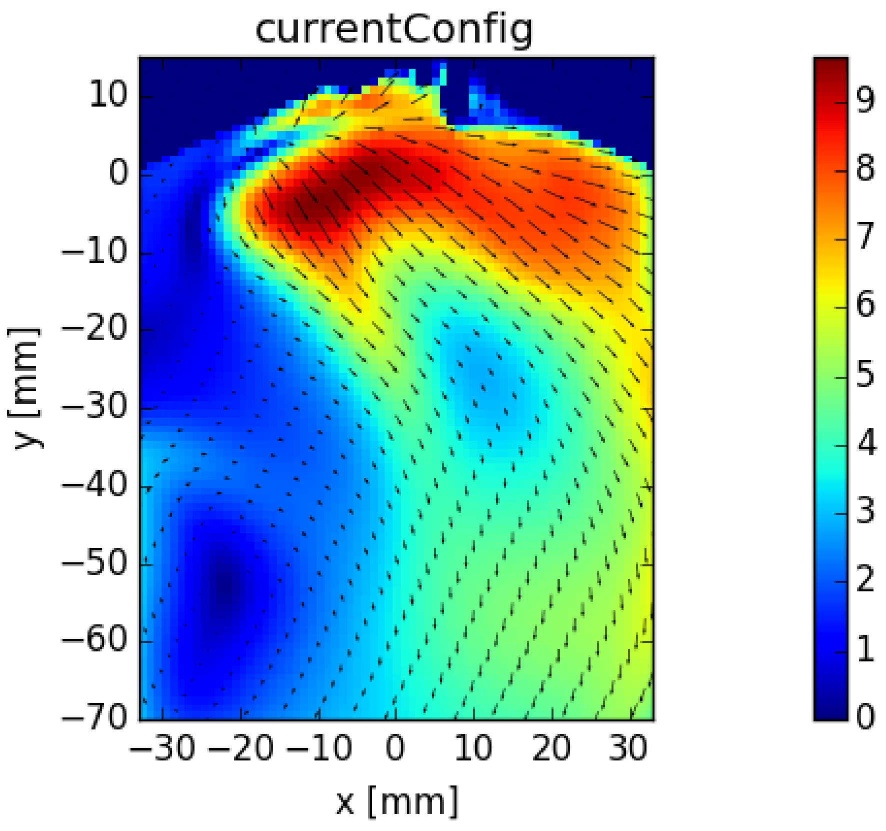


Figure 6.14: Computed velocity field on the symmetry plane at CA 540.

It can be observed that the Tumble motion does not follow the expected evolution. The circular vortex begins to appear in the bottom part of the cylinder, near the piston, shifted on the left ($-25 < x < -15$, $-50 < y < -60$ mm). However it should be already more defined and centred on the cylinder axis. Figure (6.15) shows the experimental and computed velocity flow field in the standard configuration.

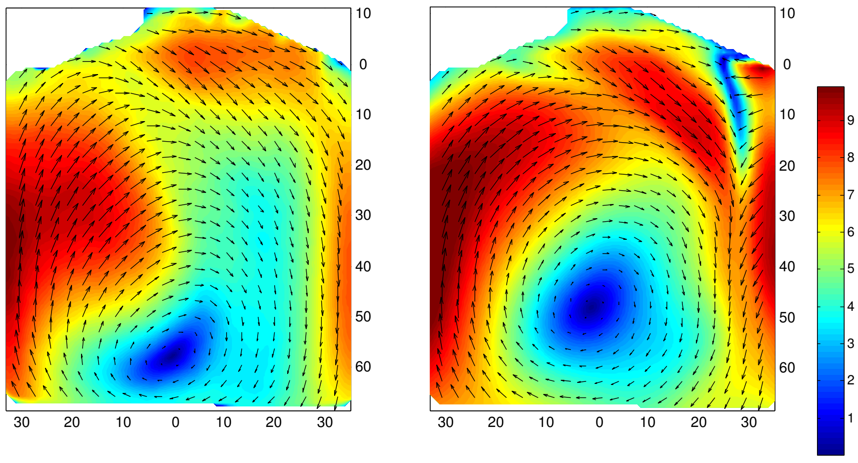


Figure 6.15: Comparison between experimental (left) and computed (right) velocity field on the symmetry plane at CA 540, taken from [16].

Once again it appears clearly that in the standard engine configuration the Tumble motion is more enhanced. In the current configuration the Tumble vortex evolution appears reduced and delayed: responsible for this effect could be the combination of a weaker direct vortex and, above all, a more intense counter-vortex with respect to the standard configuration, as highlighted in the results at CA 450.

The computed velocity components extracted along horizontal lines at different distances from the cylinder head are shown in figures (6.16) and (6.17), allowing a better quantitative comprehension of the observed phenomena.

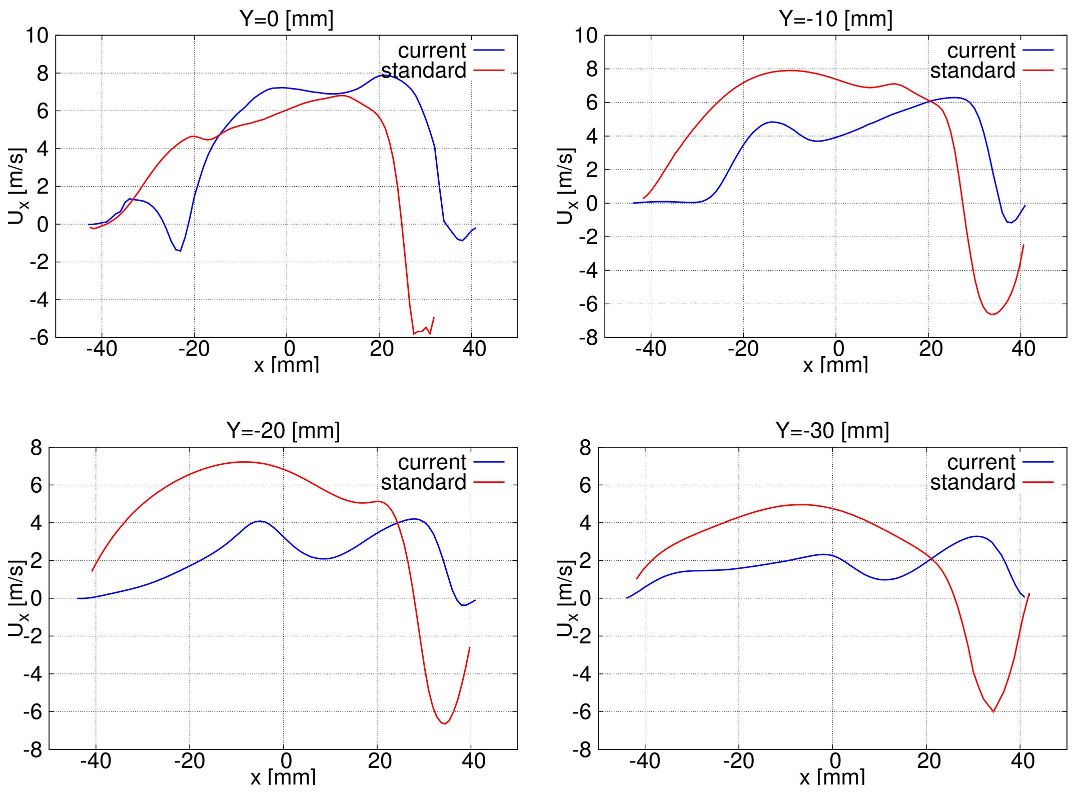


Figure 6.16: Computed velocity x-component profiles at CA 540, for both standard and current configurations.

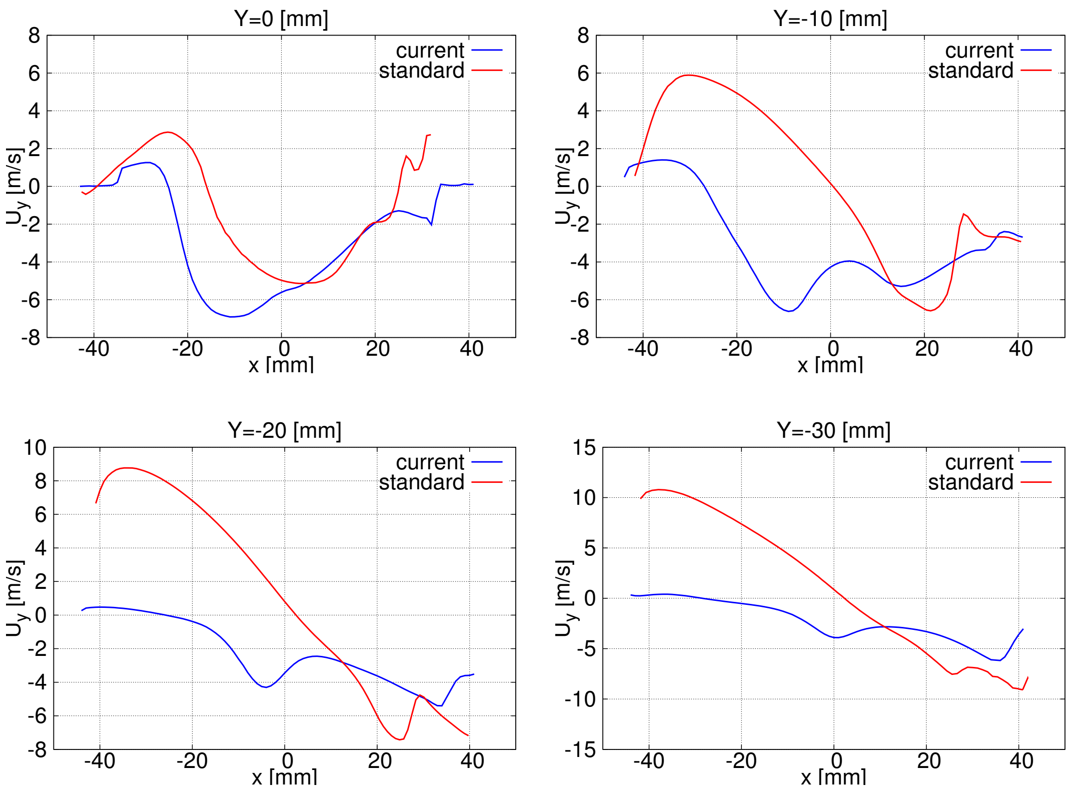


Figure 6.17: Computed velocity y-component profiles at CA 540, for both standard and current configurations.

Concerning the x-component it can be observed that at each distance from the cylinder head the curves have the same sign in the intake side of the cylinder ($x < 0$): this means that both the simulations predict a vortex moving towards right ($U_x > 0$), rotating around the negative z-axis. The velocity values however are higher in the standard configuration, since the vortex is more developed. In the exhaust side of the cylinder ($x > 0$), close to the cylinder head ($Y = 0$ and $Y = -10$ mm) the trends are comparable but in the current configuration the flow is more headed to the cylinder liner while in the standard one is already undergoing the tumble motion, thus reaching negative values. This trend is increased at $Y = -20$ and $Y = -30$mm, due to the proximity of the vortex center located along the cylinder-axis ($x = 0$) at $-60 < y < -45$ mm.

Analysing the y-component, it can be observed that close to the cylinder head ($Y = 0$ mm) the trends are comparable, while increasing the distance from the cylinder head ($Y = -10$, $Y = -20$ and $Y = -30$ mm) the y-components have opposite signs for $x < 0$: in the standard configuration the flow is moving upwards ($U_y > 0$), while in the current one the vortex is still not strong enough, thus at the sampling positions the flow is already moving downwards ($U_y < 0$). For $x > 0$ the differences are not as sharp as in the other side of the cylinder: in both the configurations the flow is going downwards, the piston will then force the flow to invert direction (in the intake side).

Figure (6.18) compares the evolution of Tumble motion from CA 450 (first row) to CA 540 (second row) in the two configurations, using the measurements on the standard configuration as a reference.

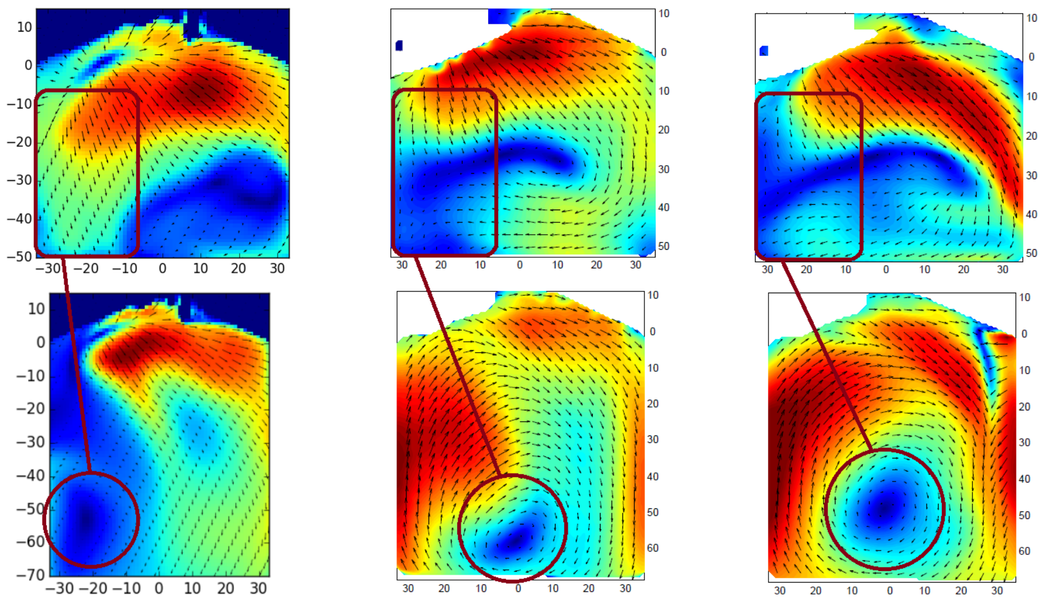


Figure 6.18: Velocity field on the symmetry plane at CA 450 (upper row) and 540 (lower row). First column: numerical results on the current configuration; second and third columns: measurements and numerical results respectively on the standard configuration.

As highlighted in figure (6.18), the different evolution of the Tumble vortex at CA 540 seems related to the intensity of the counter-vortex at CA 450. Using the measurements on the standard configuration (central column) as a reference, it can be observed that at CA 450 the counter-vortex is too intense in the current configuration (first column), due to the different geometry and to a further overestimation in the numerical results. On the other hand, the counter-vortex is underestimated in the numerical results on the standard configuration (third column). As a consequence, at CA 540 the Tumble vortex is remarkably delayed in the current simulation with respect to the measurements on the standard one. In the measurements the Tumble vortex is easy to distinguish, its center is along the cylinder-axis ($x = 0$) but still close to the piston ($y = -60$ mm). In the current configuration instead the vortex is not fully developed and its center is shifted to the left ($x = -20$ mm). In the numerical results on the standard configuration on the contrary, the direct vortex has not been obstructed by the counter-vortex and this leads to an overestimation of the Tumble evolution: the vortex center is along the cylinder-axis, but already far from the piston ($y = -50$ mm) and the velocity magnitude is higher.

Figure (6.18) shows thus the great influence of the counter-vortex in the Tumble generation and evolution, its effect must be taken into account during the design of the intake port.

CA 630, mid-compression

At CA 630 the experimental data on the current configuration are not available. Similarly to the analysis at CA 540, a comparison between current and standard configurations is thus performed.

At CA 630 the valves are closed and the piston is moving upwards, increasing the Tumble motion. In the current configuration the Tumble motion is still not completely developed, its center is moving upwards and towards the right side of the cylinder, as shown in figure (6.19).

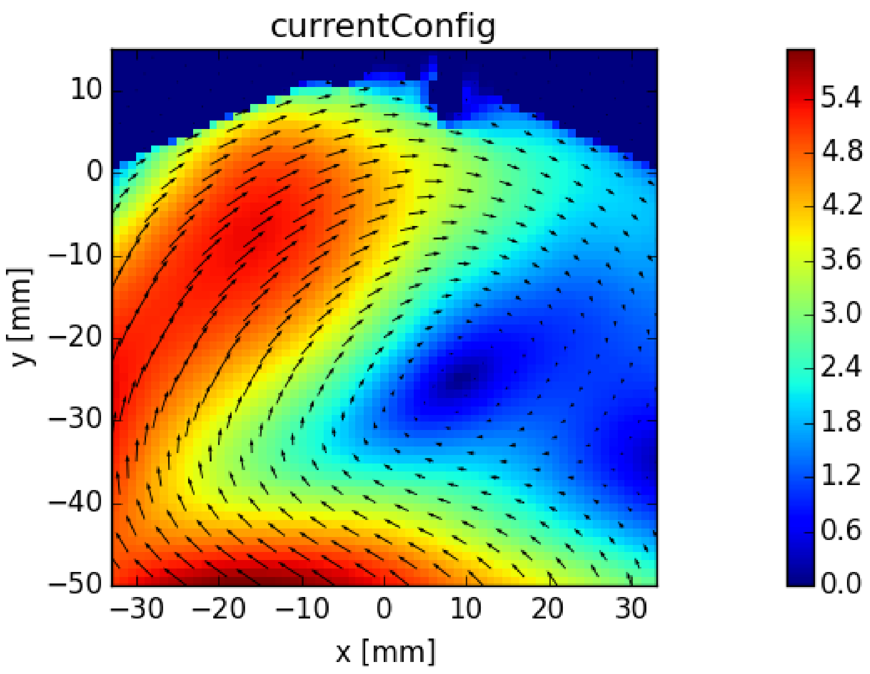


Figure 6.19: Computed velocity field on the symmetry plane at CA 630.

At mid-compression however, the Tumble center should be already on the top-right of the cylinder, as testified by the experimental data and the numerical results related to the standard engine configuration shown in figure (6.20).

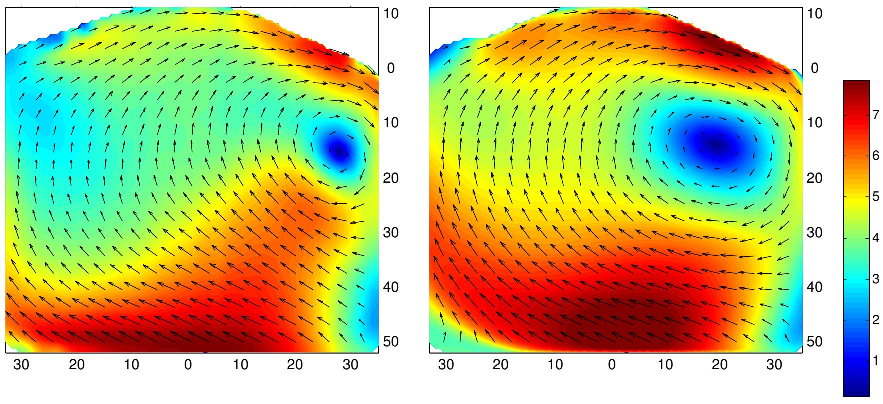


Figure 6.20: Comparison between experimental (left) and computed (right) velocity field on the symmetry plane at CA 630, taken from [16].

The computed velocity field of [16] correctly predicts the position of the vortex center, but slightly overestimates the velocity magnitude. The reason may lie on the underestimation of the counter-vortex, as highlighted above.

The computed velocity components extracted along horizontal lines at different distances from the cylinder head are shown in figures (6.21) and (6.22).

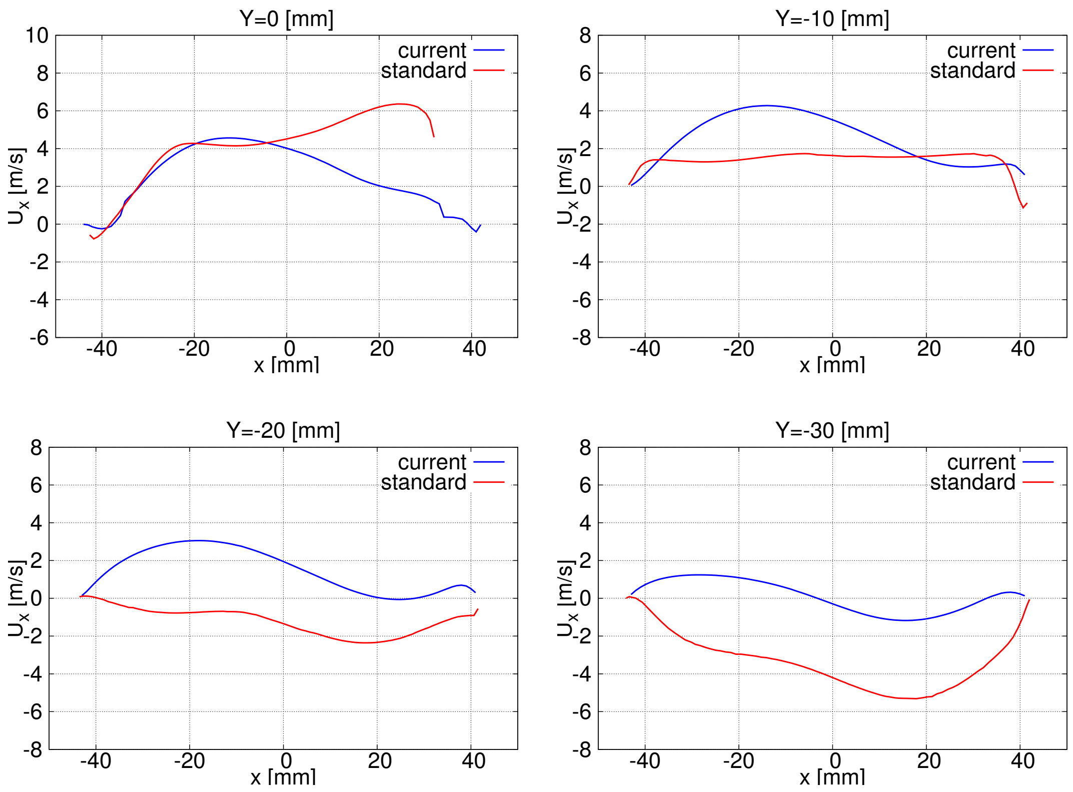


Figure 6.21: Velocity x-component profiles at CA 630, for both standard and current configurations.

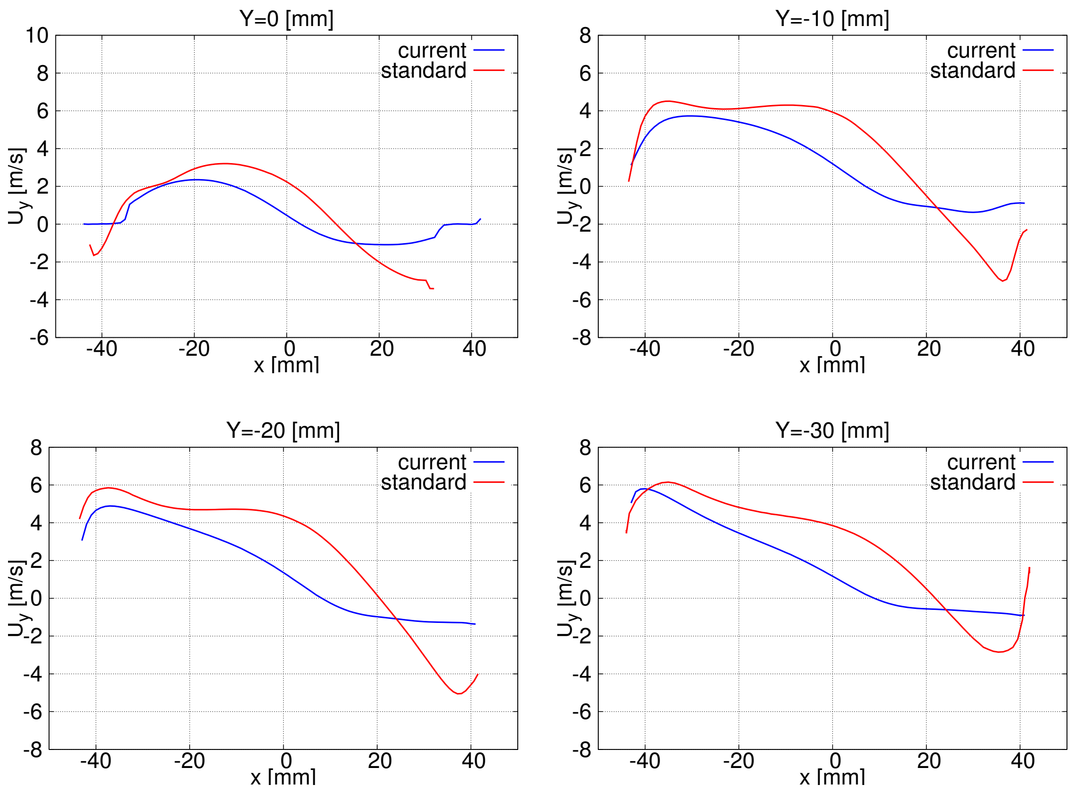


Figure 6.22: Velocity y-component profiles at CA 630, for both standard and current configurations.

The velocity components trends are not as dissimilar as at CA 540. The Tumble vortex is now clearly identifiable also in the current simulation, even if it is still not fully developed.

Close to the cylinder head ($Y = 0$ and $Y = -10$ mm) the trends are similar in the two configurations, higher differences take place on the exhaust part of the cylinder ($x > 0$). In the current configuration the vortex center is at $x = 10$, $y = -25$ mm, still not at the top-right of the combustion chamber as in the standard one ($20 < x < 30$, $y = -15$ mm). Moreover the radius of rotation in the current configuration is much bigger than in the standard one, the flow does not undergo a clear circular rotation. For $x > 10$ mm the slope of the y-component curve representing the current configuration is thus not as high as the one representing the standard configuration, where the circular vortex in fully developed.

At higher distance from the cylinder head instead ($Y = -20$ and $Y = -30$ mm), the sign of the x-component of the velocity is opposite in the two cases. In the current configuration the vortex center is located between these two sampling lines, thus these samplings catch the upper part of the vortex, moving towards right

($U_x > 0$). On the contrary, in the standard configuration the vortex center is located at $y = -15$ mm, thus these samplings catch the lower part of the vortex, moving towards left ($U_x < 0$). Concerning the y-component the differences decrease except for $x > 20$ mm: here in the standard configuration the flow is moving downwards with higher velocity with respect to the current configuration, because the vortex center is at $20 < x < 30$ mm and the rotational radius is of the order of 5-10 mm.

6.2.2 Cylinder pressure trend

An important check for the validity of the numerical setup, the geometry discretization and the imposed boundary conditions is represented by the cylinder pressure trend in the CA-range where the computed velocity flow field has been analysed. Figure (6.23) shows the computed and the experimental in-cylinder pressure in the first part of the engine cycle.

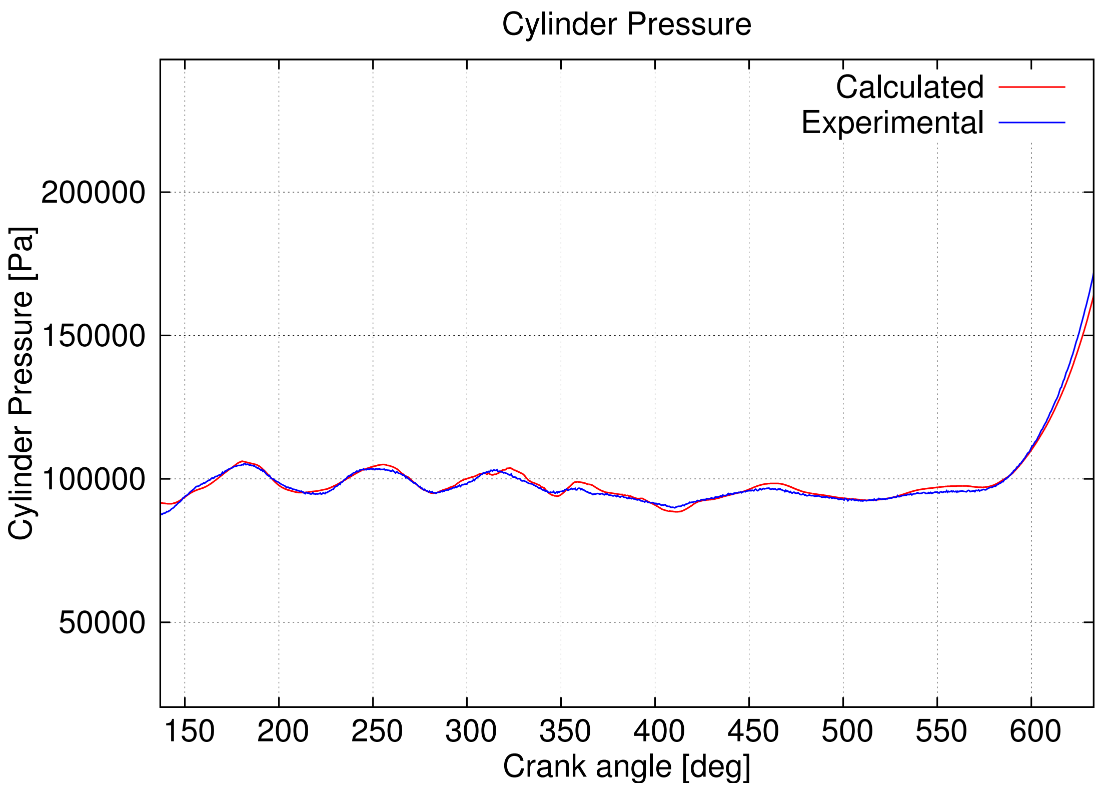


Figure 6.23: Comparison between computed and experimental cylinder pressure trend in the current configuration - zoom on the CA-range of interest.

As it possible to observe, the computed and the experimental cylinder pressure trends are in good agreement. The computed flow field in this CA range is thus not affected by numerical issues.

During the compression phase however, the computed cylinder pressure profile differs remarkably from the experimental one, as shown in figure (6.24).

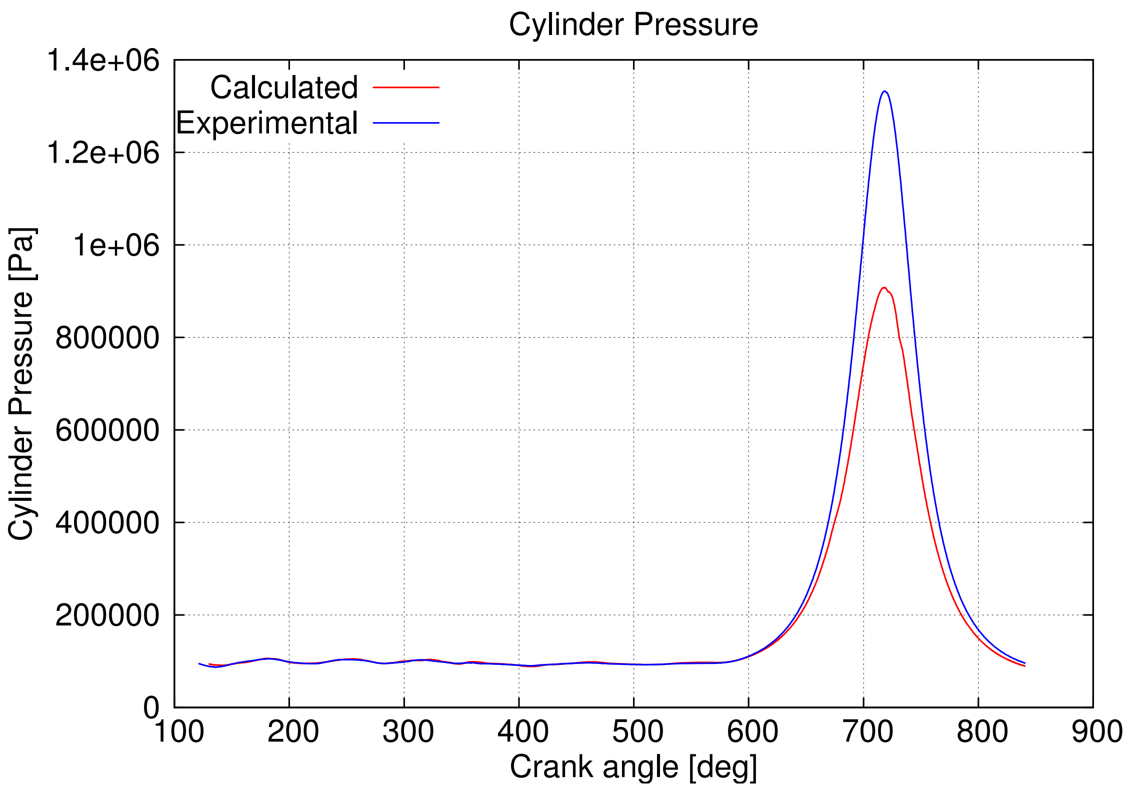


Figure 6.24: Comparison between computed and experimental cylinder pressure trend in the current configuration - full pressure profile.

The pressure peak is correctly located, but the peak value is much lower in the computed results, with a resulting compression ratio decreased of about 2 unity. This decrement of the pressure peak is due to the combustion chamber crevice modeling: the compression ratio is strongly affected by the addition of the crevices[1]. Their presence has thus to be taken into account not only for their great influence on the thermal efficiency and the pollutants production, but also for their effect on the compression ratio. Their design can thus be optimized in order to adjust the compression ratio and the engine-out hydrocarbon emissions. In the present case, their effect on the cylinder pressure during the compression phase leads to uncertainty in

[1]previous simulation performed on the current engine configuration without modeling the combustion chamber crevices had computed the correct pressure peak.

the prediction of the flow field in the final part of the engine cycle. Moreover with the achieved pressure peak a combustion process would not be correctly analysed.

Conclusions

In this work the gas-exchange in the two configurations of the TU-Darmstadt optical engine has been analysed by means of CFD modeling. Two types of simulations were addressed: the steady-state simulation of the mid-intake phase and the full-cycle simulation.

The aim of the steady-state simulation was first the experimental validation on the standard engine configuration. Two types of sensitivity analysis have been performed: turbulence model and numerical schemes. Comparing the results achieved using the k-ε and the k-ω SST models, the best agreement with the experimental data has been obtained with the k-ε model, proving its reliability in the analysis of complex geometry. Concerning the sensitivity analysis of the numerical schemes, it has been proved that in steady-state condition improving the accuracy leads to best agreement with experimental data, without compromising stability.

Then a comparison between the two engine configurations has been carried out. The standard configuration has demonstrated a better capability to generate a structured Tumble motion with respect to the current one. This result was expected: the orientation of the intake ports in the standard configuration drives the incoming flow toward the liner on the exhaust side of the combustion chamber, while in the current configuration the intake duct ends more uprightly.

Performing the full-cycle simulation required a greater effort. This type of simulation has great potentiality for the investigation of the complex flow and thermal phenomena taking place in internal combustion engines. A good prediction of the flow motion, the fuel-air mixing and the overall in-cylinder conditions is necessary to correctly predict and optimize the combustion process and the pollutant production. The optimization of the automatic methodology allowing to perform this type of simulation is thus of central importance and interest. For this reason the entire procedure, from the generation of the full-cycle set of meshes to the implementation of the settings required by the applications, has been described and explained in detail.

The full-cycle simulation has been performed on the current engine configuration only. The quality of the computational grid and the applied settings have been validated by comparing the numerical results with the experimental data available at mid-intake (CA 450). The achieved agreement can be regarded as good: it results in line with the agreement achieved in previous work on the standard configuration. It is remarkable however, the overestimation of the counter-vortex: this overestimation leads to a delayed and reduced evolution of the Tumble motion in the following engine phase. This problem can be overcome by improving the flow-grid alignment in the intake valve region and further reducing the grid resolution.

Further comparisons have been carried out between the results achieved on the current configuration in the present work and the results taken from a previous work of the ICE Group of Politecnico di Milano on the standard configuration. The Tumble motion has been analysed at CA 450 (mid-intake), 540 (intake, piston at BDC) and 630 (mid-compression), allowing to observe the Tumble generation and evolution. Thanks to the experimental data at CA 450 available for both configurations, it has been possible to verify that the difference in the numerical results was due mainly to the different geometry. The further comparisons on the following engine phase were thus meaningful from a motoring point of view. The full-cycle analysis has confirmed the result of the steady-state simulation: the standard configuration generates a higher Tumble motion with respect to the current one.

The full cycle simulation allows to analyse also the cylinder pressure. If the velocity field is of central importance for a good fuel-air mixing and to achieve a high turbulence level inside the combustion chamber, also the achieved pressure during the compression phase plays an important role in the optimization of the combustion process. In the present work, the peak of in-cylinder pressure has not been correctly predicted: it is consistently lower than the experimental peak. For this reason a following combustion process simulation would not be possible, as well as a correct prediction of the flow field during the final part of the compression phase. The cause of this wrong prediction of the pressure peak lies in the combustion crevice modeling, which has increased the in-cylinder volume leading to a remarkable decrease of the compression ratio. The encountered problem however demonstrates that the design of the combustion chamber crevice has to be taken into account not only for the thermal efficiency and the pollutant production, but also for the control of the compression ratio.

Further analyses may be carried out in future works under full-cycle conditions. First of all a combustion process simulation taking as input the in-cylinder conditions achieved by the cold-flow simulation. Another interesting analysis would be the investigation of the effect of the imposed symmetry condition, which allows to perform the simulation on half the engine. The geometrical symmetry of the domain does not ensure a symmetric evolution of the in-cylinder flow field. The imposed symmetry condition allows to strongly reduce the computational costs still preserving a reasonable good phenomena prediction, but studying the entire domain could lead to different results especially on the symmetry plane, due to the interaction between the two incoming flows.

Acronyms

BDC Bottom Dead Center.

bTDC before Top Dead Center.

CA Crank-Angle Degree.

CAD Computer-Aided Design.

CAE Computer-Aided Engineering.

CFD Computational Fluid Dynamics.

CV Control Volume.

DNS Direct Numerical Simulation.

EVC Exhaust Valve Closing.

EVO Exhaust Valve Opening.

FDM Finite Difference Method.

FEM Finite Element Method.

FGA Flow Grid Alignment index.

FVM Finite Volume Method.

GDI Gasoline Direct Injection.

ICE Internal Combustion Engine.

IVC Intake Valve Closing.

IVO Intake Valve Opening.

LES Large Eddy Simulation.

MRV Magnetic Resonance Velocimetry measurement.

OBJ Wavefront Object format.

OpenFOAM Open Source Field Operation and Manipulation.

PDE Partial Differential Equation.

PISO Pressure Implicit with Splitting of Operators Algorithm.

PIV Particle Image Velocimetry measurement.

R&D Research and Development.

RANS Reynolds-averaged Navier–Stokes Equations.

SI Spark-Ignition.

SIMPLE Semi-Implicit Method for Pressure Linked Equations Algorithm.

STL Stereolithography format.

TDC Top Dead Center.

Bibliography

[1] F. G. L. Amorim, J. H. M. Ribeiro, M. G. J. Vaz, R. M. Valle, *Sensitivity Analysis of the Air Flow inside a Single Cylinder Engine for Different Turbulence Models Using CFD*, Advanced Materials Research, Vol. 1016: 624-629, 2014.

[2] ANSYS FLUENT website, *Theory Guide*

[3] C. Arcoumanis, S. Godwin, J. Kim, *Effect of Tumble Strength on Combustion and Exhaust Emissions in a Single-Cylinder, Four-Valve, Spark-Ignition Engine*, SAE Technical Paper, 981044, 1998.

[4] C. Berggren, T. Magnusson, *Reducing automotive emissions - The potentials of combustion engine technologies and the power of policy*, Energy Policy, Vol. 41:636-643, 2012.

[5] A. Della Torre, T. Lucchini, G. D'Errico, G. Montenegro, M. Fiocco, A. Maghbouli, *Full-cycle simulation of the Darmstadt engine*, Third Darmstadt engine workshop, 2015.

[6] P. Domingo, L. Vervisch, J. Reveillon, *DNS analysis of partially premixed combustion in spray and gaseous turbulent flame-based stabilized in hot air*, Combustion and Flame, Vol. 140(3):172-195, 2005.

[7] S. Falfari, F. Brusiani, G. Bianchi, *Assessment of the Influence of Intake Duct Geometrical Parameters on the Tumble Motion Generation in a Small Gasoline Engine*, SAE Technical Paper, 2012-32-0095, 2012.

[8] G. Ferrari, *Motori a combustione interna*, Il Capitello, Torino, 4^a edizione, 2008.

[9] M. Fiocco, T. Lucchini, *Guidelines to the engineDynamicSetUp application*

[10] M. Fiocco, T. Lucchini, *Guidelines to case setup for a gas exchange simulation of IC engine geometries*

[11] D. Freudenhammer, E. Baum, B. Peterson, B. Böhm, B. Jung, S. Grundmann, *Volumetric intake flow measurements of an IC engine using magnetic resonance velocimetry*, Experiments in Fluids, 55:1724, 2014.

[12] A. D. Gosman, *State of the Art of Multi-Dimensional Modeling of Engine Reacting Flows*, Oil & Gas Science and Technology, Vol. 54-2: 149-159, 1999.

[13] R.F. Huang, C.W. Huang, S.B. Chang, H.S. Yang, T.W. Lin, W.Y. Hsu, *Topological flow evolutions in cylinder of a motored engine during intake and compression strokes*, Journal of Fluids and Structures, Vol. 20(1):105-127, 2005.

[14] H. Jasak, Z. Tukovic, *Automatic mesh motion for the unstructured finite volume method*, Transactions of FAMENA, XXX-2, 2006.

[15] P. K. Kundu,I. M. Cohen, D. R. Dowling, *Fluid Mechanics*, Academic Press, 6^{th} edition, 2015

[16] T. Lucchini, A. Della Torre, G. D'Errico, G. Montenegro et al., *Automatic mesh generation for CFD simulations of direct-injection engines*, SAE Technical Paper, 2015-01-0376, 2015.

[17] T. Lucchini, G. D'Errico, H. Jasak, Z. Tukovic, *Automatic mesh motion with topological changes for engine simulation*, SAE Technical Paper, 2007-01-0170, 2007.

[18] T. Lucchini, M. Fiocco, R. Torelli, G. D'Errico, *Automatic mesh generation for full-cycle CFD modeling of IC engines: application to the TTC test case*, SAE Technical Paper, 2014-01-1131, 2014.

[19] F. Moukalled, L. Mangani, M. Darwish, *The Finite volume method in computational fluid dynamics -an advanced introduction with OpenFOAM and Matlab* Springer, 2016.

[20] OpenFOAM website, *User Guide*

[21] S.B. Pope, *Turbulent Flows*, Cambridge University Press, 2000.

[22] O. Reynolds, *An experimental investigation of the circumstances which determine whether the motion of water shall be direct or sinuous, and of the law of resistance in parallel channels*, Phil. Trans. R. Soc., 174:935-982, 1883.

[23] S. Richard, O. Colin, O. Vermorel, A. Benkenida, C. Angelberger, D. Veynante, *Towards large eddy simulation of combustion in spark ignition engines*, Proceedings of the Combustion Institute, 31 3059-3066 , 2007.

[24] L. F. Richardson, *Weather prediction by numerical process*, Cambridge University Press, 1922.

[25] O. Vermorel, S. Richard, O. Colin, C. Angelberger et al., *Multi-Cycle LES Simulations of Flow and Combustion in a PFI SI 4-Valve Production Engine*, SAE Technical Paper, 2007-01-0151, 2007.

[26] O. Vermorel, S. Richard, O. Colin, C. Angelberger, A. Benkenida, D. Veynante, *Towards the understanding of cyclic variability in a spark ignited engine using multi-cycle LES*, Combustion and Flame, Vol. 156(8):1525-1541, 2009.

[27] H. K. Versteeg, W. Malalasekera, *An introduction to computational fluid dynamics: the finite volume method*, Pearson Education, 2^{nd} edition, 2007.