OpenFOAM®

Simulation of the first stages of a turbofan using OpenFOAM

Report

Degree: Aerospace Engineering
Course: Application of Open-Source CFD to Engineering Problems
Delivery date: 09-12-2016

Students:

Herrán Albelda, Fernando
Martínez Viol, Víctor
Morata Carranza, David

Contents

1 Introduction

During the development of the course *'Application of Open-Source CFD to engineering problems'* we have learned the basics of how to use and solve real-world cases and situations related with fluid mechanics using OpenFOAM, an open source CFD tool. The first days of the course, several possible projects were presented and we had to choose one of them and make a report. After agreeing with the professor, we decided to simulate an option that was not on that list. Since we are really interested in propulsion, we thought that it was a good idea to try to simulate the flow inside the first stages of compression of a turbofan engine.

These type of engines are the most used propulsion system in the aerospace industry. It presents several advantages to other systems such as the turbohelix or the turbojet; for example, this kind of engine takes advantage of the flow that goes through the fan (that cannot be more compressed or heated given that the combustion chamber has a limited volume) and comes out through the rear nozzle. This results in a higher thrust for the same amount of fuel that is burned. Thus, it is no surprise that state-of-the-art planes such as the Airbus 380 or the Boeing 747 use this kind of propulsion system.

In order to do a realistic simulation, we have been gathering lots of information of this type of engines and their typical working conditions. Also, we have been searching for information about how could we solve a geometry that is rotating. Several simulations and comparisons will be presented in this report to analyze the validity of the results obtained.

It will be presented in this report how to use the Arbitrary Mesh Interface utility (AMI) as well as the boundary conditions, the mesh generation and the solver that has been used for the case. Additionally, several hypotheses will be considered in order to alleviate the computation time (given that this project has been simulated in a laptop).

2 Case

2.1 Description of the case

The aim of the current project is to analyze the airflow inside the first stages of a turbofan. The turbofan consists of an initial stage where a fan is placed. This first stage increases the pressure of the air that goes through it and, as it can be seen in 2.1, a part of the incoming airflow goes to the low pressure compression stage (main or primary flow) and the other goes to the conducts (secondary flow). As the secondary flow advances through the conduct, the main flow enters the engine core where it goes through the low pressure and high pressure compression stages. By increasing the pressure of the air, the density also increases; thus, a higher mass flow can be mixed with the fuel and burned in the combustion chamber. After this process, the gases go through the turbines, interchanging energy with them (pressure and velocity, mainly), and they arrive at the nozzle, where the air is accelerated, and a thrust force is obtained.

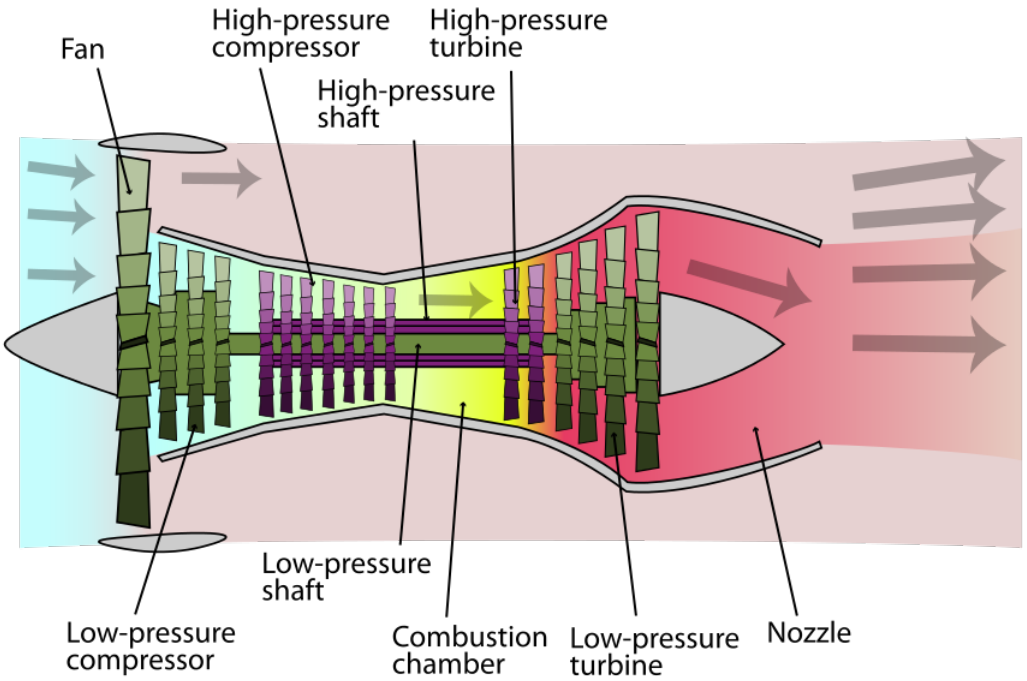

Figure 2.1: Parts of a turbofan

To simulate the flow in the first stages of the turbofan, we have downloaded the following turbine model from *GrabCad*, shown in 2.2. This model is pretty similar with the one shown above (2.1) and the simulation will take place between the back side of the fan and the back side of the second compressor. There are several realistic and potentially functional models on *GrabCad*; however, this model has been selected because it did not present incompatibilities with *SolidWorks* and *Salome*. Given that all the parts of the model have to be clearly differentiated in order to export them as *STL* files, we could not use an assembly.

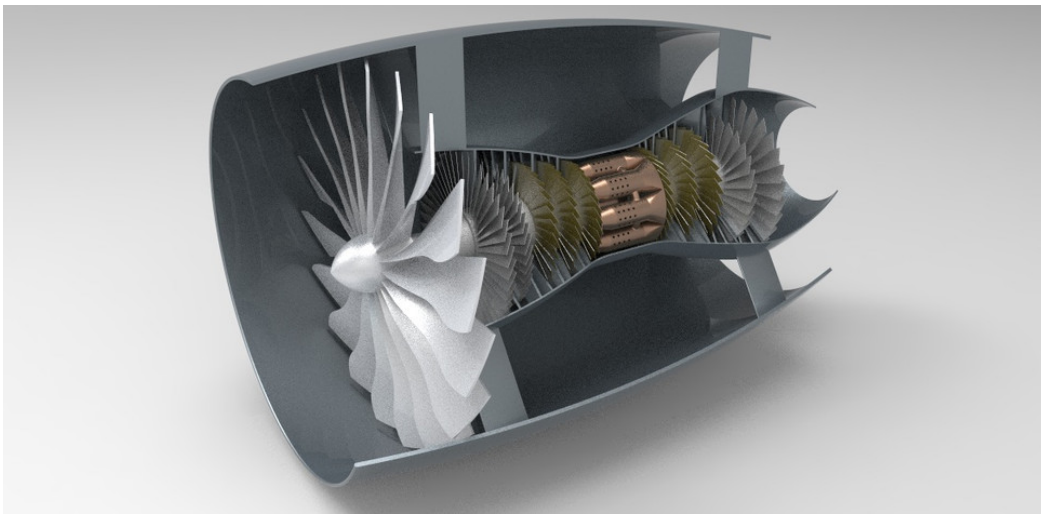

Figure 2.2: Model used

2.2 Hypotheses

Given that we are not able to use a supercomputer and the computational power that is available to us is very limited since we are using our own personal computers to run this simulation, several hypotheses have to be made to alleviate the calculation time, as mentioned before. Some of them can be assumed and some other will make the case a non-realistic project. These hypotheses are presented below.

- Incompressible flow
- Newtonian flow

- Laminar flow

Clearly, the flow is not incompressible in reality; the aim of the turbofan is to compress the air to seek for a more efficient combustion. However, all of the compressible solvers are really difficult to use and the computational time is also higher given that we have a high number of control volumes (as discussed in the next section) due to a really complex geometry. Thus, although it will not be a completely real flow, we will be also able to see how the low pressure section of the engine increases the pressure of the flow and how the velocity field changes as it goes through the turbine.

The simulation will take into account that it is a tridimensional flow, also that it behaves as a newtonian fluid and it will be run under transient flow conditions (that is: the velocity field as well as the values of the pressure change over time). With these hypothesis, we can begin to discuss the geometry of the *blockMesh* as well as the refined mesh, and the boundary conditions and final results in the next sections.

Finally, given that it is a fictional case, no validations of the solution will be performed.

3 Pre-processing

3.1 Mesh generation

3.1.1 Selection of the tutorial

The selection of the tutorial case is the first thing that needs to be done to run the simulation of the turbofan since all the files that will be used are linked and, starting a new case would mean spending a lot of time creating files and establishing all these links in order to make *OpenFOAM* work properly. Among all the tutorials that can be found on the *OpenFOAM* folder, we have selected the *mixerVesselAMI2D* for several reasons.

3.1.2 blockMesh

The definition of the *blockMeshDict* is the first part that needs to be modified. The *blockMesh* must contain the geometry that has to be simulated and we must have an idea of the vertices of the parallelogram that will contain the first stages of the compressor. In order to do that, the geometry has to be opened using either *Salome* or *Paraview*. Then the axes must be showed and the points have to be written down on the *blockMesh* file.

Now that the points enclosing the desired geometry are clear, they can be modified in the `system/blockMeshDict` file. This change is presented below:

```
vertices
(
    (0.77 0   0)
    (1.35 0   0)
    (1.35 2.3 0)
    (0.77 2.3 0)
    (0.77 0   2.3)
    (1.35 0   2.3)
    (1.35 2.3 2.3)
    (0.77 2.3 2.3)
);
```

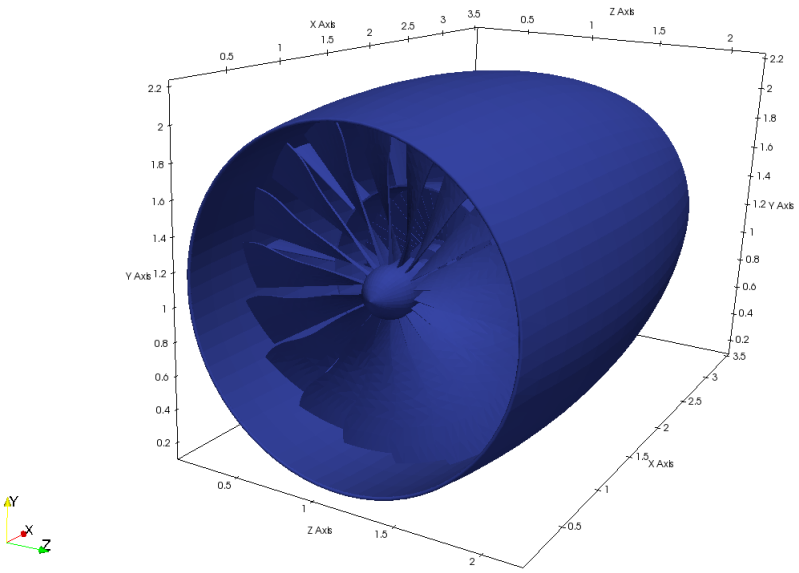

Figure 3.1: Turbofan with axes

It can be clearly seen that the domain of the mesh is a 0.58x2.3x2.3m rectangular prism.

Once the boundaries of the *blockMesh* are defined, the number of cells that it will have has to be set. It has to be taken into account that a very dense mesh at the beginning will not be efficient when simulating the case given that we will use the *snappyHexMesh* later, so it might be over densified. On the other hand, a very coarse mesh will not be efficient either because additional divisions will have to be set when generating the refined mesh and the computational time of the *snappyHexMesh* process might grow. Thus, a solution between a mesh with a very high number of cells and a very low number of cells has to be attained.

This basic mesh has been divided every 0.05m. It means that we have done 12 divisions in the x direction, 46 on the y direction and 46 more on the z direction.

```
blocks
```

```
(
    hex (0 1 2 3 4 5 6 7) (12 46 46) simpleGrading (1 1 1)
);
```

Finally, the differents faces of the mesh must be defined depending on whether they are the inlet or outlet faces or the lateral faces of the geometry. To do this, *OpenFOAM* numbers the vertices according to their appearance in the *blockMeshDict* and the faces are defined by four those numbers. Since the vertex numeration has been kept the same as the one that comes as default in every tutorial case; it is easier to define the inlet face of the *blockMesh* (this is: the face in which the flow comes in) as well as the outlet face (this is: the face in which the flow comes out) that will be used to define the boundary conditions later.

```
boundary
(
    frontAndBack
    {
        type patch;
        faces
        (
            (3 7 6 2)
            (1 5 4 0)
            (0 3 2 1)
            (4 5 6 7)
        );
    }
    inlet
    {
        type patch;
        faces
        (
            (0 4 7 3)
        );
    }
    outlet
    {
        type patch;
        faces
```

```
        (
            (2 6 5 1)
        );
    }
);
```

The *blockMeshDict* file is presented below. This is the final file that has been used to generate the *blockMesh* and it is included in the *.zip* file attached to this report. Only the parameters mentioned above have been modified; the rest is equal to the tutorial case that has been selected.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.0                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}

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

convertToMeters 1;

vertices
(
    (0.77 0   0)
    (1.35 0   0)
    (1.35 2.3 0)
    (0.77 2.3 0)
    (0.77 0   2.3)
    (1.35 0   2.3)
    (1.35 2.3 2.3)
    (0.77 2.3 2.3)
```

```
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (12 46 46) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
    frontAndBack
    {
        type patch;
        faces
        (
            (3 7 6 2)
            (1 5 4 0)
            (0 3 2 1)
            (4 5 6 7)
        );
    }
    inlet
    {
        type patch;
        faces
        (
            (0 4 7 3)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (2 6 5 1)
        );
    }
);
```

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

The log obtained when the *blockMesh* has been generated is presented below. As it can be seen, no errors were found during the computation of this basic mesh. The number of cells is relatively high (**25392**) but perfectly suitable to proceed with the refinement of the mesh. Additionally, a caption of the basic mesh is presented in 3.2.

```
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.0                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build  : 4.0-665f1db4c1f1
Exec   : blockMesh
Date   : Dec 05 2016
Time   : 14:17:05
Host   : "victorPC"
PID    : 4499
Case   : /home/victor/Desktop/Final/pimpleDyMFoam/turbofan_std
nProcs : 1
sigFpe : Enabling floating point exception trapping (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using timeStampMaster
allowSystemOperations : Allowing user-supplied system call operations

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

Creating block mesh from
    "/home/victor/Desktop/Final/pimpleDyMFoam/turbofan_std/system/blockMeshDict"
Creating curved edges
Creating topology blocks
Creating topology patches

Creating block mesh topology

Check topology
```

```
Basic statistics
Number of internal faces : 0
Number of boundary faces : 6
Number of defined boundary faces : 6
Number of undefined boundary faces : 0
Checking patch -> block consistency

Creating block offsets
Creating merge list .

Creating polyMesh from blockMesh
Creating patches
Creating cells
Creating points with scale 1
    Block 0 cell size :
        i : 0.0483333 .. 0.0483333
        j : 0.05 .. 0.05
        k : 0.05 .. 0.05


There are no merge patch pairs edges

Writing polyMesh
----------------
Mesh Information
----------------
  boundingBox: (0.77 0 0) (1.35 2.3 2.3)
  nPoints: 28717
  nCells: 25392
  nFaces: 79396
  nInternalFaces: 72956
----------------
Patches
----------------
  patch 0 (start: 72956 size: 2208) name: frontAndBack
  patch 1 (start: 75164 size: 2116) name: inlet
  patch 2 (start: 77280 size: 2116) name: outlet

End
```

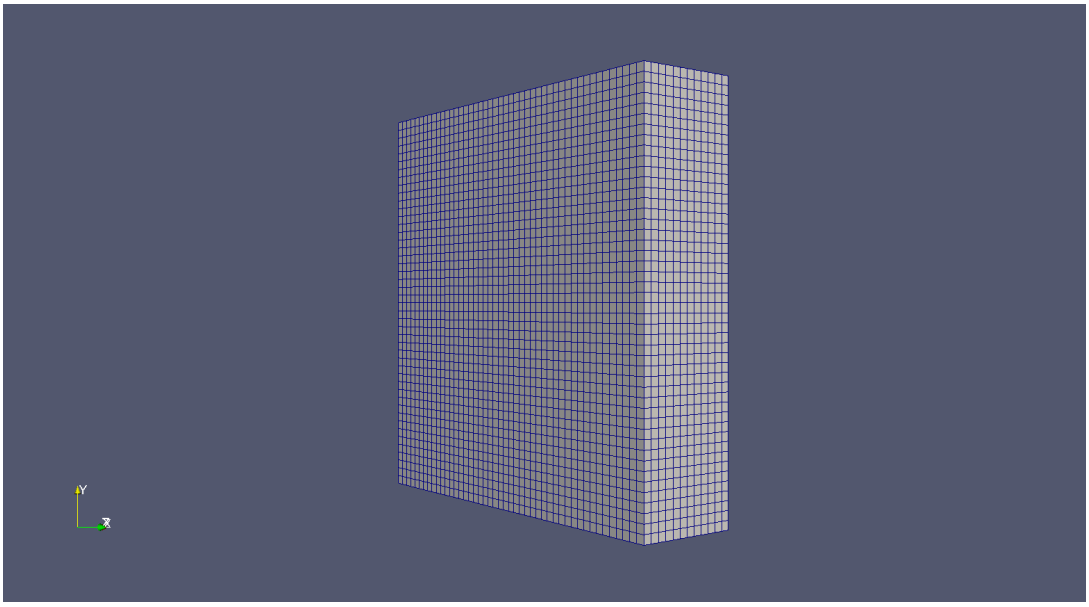

Figure 3.2: blockMeshcaption

3.1.3 Mesh refinement

To refine the mesh, the *snappyHexMesh* utility is used (included in *OpenFOAM*) and several parameters have to be modified in order to obtain a dense mesh that is suitable for the simulation of this complex geometry. It has to be considered that a particular geometry is has a relative velocity as well; this is, the rotor is rotating, and so the first and second stages of the Low Pressure Compression of the turbofan engine, while the nacelle and the combustor are static.

In the `system/snappyHexMeshDict` several parameters have to be modified. First, the *snappyHexMesh* must be aware of the geometries that it has to take into account. As it can be seen below, the files *Fan.stl*, *HPSpool.stl*, *LPSpool.stl* and *NacelleStator.stl* have been included here.

```
geometry
{
    box1x1x1
    {
        type searchableBox;
        min (1.5 1 -0.5);
```

```
        max (3.5 2 0.5);
    }

    Fan.stl
    {
        type triSurfaceMesh;
  name Fan;
    }

    HPSpool.stl
    {
        type triSurfaceMesh;
name HPSpool;
    }

    LPSpool.stl
    {
        type triSurfaceMesh;
name LPSpool;
    }

    NacelleStator.stl
    {
        type triSurfaceMesh;
name NacelleStator;
    }

    AMI.stl    //used for the dynamic mesh (the domain where the mesh  rotating)
    {
        type        triSurfaceMesh;
        name        AMI;
    }
};
```

Next, the number of control volumes has to be limited to ensure that the laptop is capable of running a simulation. This number has been limited to four million cells, which is a pretty high number and the following lines have to be modified. Also, the maximum local number of control volumes has been limited to two million.

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

// If local number of cells is >= maxLocalCells on any processor
// switches from from refinement followed by balancing
// (current method) to (weighted) balancing before refinement.
maxLocalCells 2000000;

// Overall cell limit (approximately). Refinement will stop immediately
// upon reaching this number so a refinement level might not complete.
// Note that this is the number of cells before removing the part which
// is not 'visible' from the keepPoint. The final number of cells might
// actually be a lot less.
maxGlobalCells 4000000;
```

The next step is to define the required minimum and maximum refinement levels of the mesh for the different geometries that will be simulated. It can be clearly seen in this section that we have included the same *STL* files that we did before. The higher the level of the refinement, the denser the mesh will be and the better it will resemble to the real geometry. But the limitation here is the computational power available so, the maximum refinement number cannot be as high as we would like to. Thus, depending on the complexity of the geometry, several minimum (the first number) and maximum (the second number) refinement levels have been defined.

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

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

refinementSurfaces
{

    LPSpool
    {
```

```
            // Surface-wise min and max refinement level
            level (3 4);
            patchInfo
            {
                type wall;
                inGroups (movingWalls);
            }
}
        NacelleStator
        {
            // Surface-wise min and max refinement level
            level (2 3);
            patchInfo
            {
                type wall;
                inGroups (staticWalls);
            }
}

        Fan
        {
            // Surface-wise min and max refinement level
            level (1 2);
            patchInfo
            {
                type wall;
                inGroups (movingWalls);
            }
}

        HPSpool
        {
            // Surface-wise min and max refinement level
            level (1 2);
            patchInfo
            {
                type wall;
                inGroups (movingWalls);
            }
}
```

```
        AMI
        {
            level (2 3);

            faceType boundary;
            cellZone innerAMI;
            faceZone innerAMI;
            cellZoneInside inside;
        }
    }
```

It is worth mentioning that new zones have been created in the file. As it can be seen above, the 'AMI' zone has been defined given that it will be the rotating zone of the turbofan. Also, besides the level of refinement for each geometry, it can be seen that it has been defined a type of group for each of them. So, the zones are now assembled into either the rotating geometry or static geometry.

```
        LPSpool
        {
            // Surface-wise min and max refinement level
            level (3 4);
            patchInfo
            {
                type wall;
                inGroups (movingWalls);
            }   //^MODIFIED PART
}
```

The next step is to select a point within the mesh. So, the *locationInMesh* has to be modified with the x, y, and z-coordinates of a point with that feature.

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

    // After refinement patches get added for all refinementSurfaces and
    // all cells intersecting the surfaces get put into these patches. The
```

```
// section reachable from the locationInMesh is kept.
// NOTE: This point should never be on a face, always inside a cell, even
// after refinement.
// This is an outside point locationInMesh (-0.033 -0.033 0.0033);
locationInMesh (.97443222 1.40534444343 1.24221211); // Inside point

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

Finally, the *surfaceFeatureExtract* has been used. What this option does it to refine even more the mesh near the points that have complex geometries such as the edges of the blades. This is particularly useful for the turbofan given that it has a high number of blades and a twisting geometry that will resemble more to the reality when using this option. Thus, the following lines within the *snappyHexMeshDict* must be modified.

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

// Specifies a level for any cell intersected by its edges.
// This is a featureEdgeMesh, read from constant/triSurface for now.
features
(
    {
       file "NacelleStator.eMesh";
       level 2;
    //    levels ((0.0 2) (1.0 3));
    }
    {
       file "LPSpool.eMesh";
       level 3;
    //    levels ((0.0 2) (1.0 3));
    }
);
```

All of the other parameters of the *snappyHexMeshDict* have not been modified.

3.1.4 Comparison of different types of mesh

A comparison between the meshes generated is presented below. Several meshes have been made in order to select the best one to perfom the simulation of the given case.

The mesh that has been used to simulate the case is the **Standard Mesh**. Since the computational power that we have available is limited, a very dense mesh would end up in a very tedious process that could take several hours in order to be simulated. Thus, given that the standard mesh is pretty good already, we have selected it because the results will also be quite similar.

The aim of this section is to show that the difference between the dense mesh and the standard mesh is relatively small. The standard mesh has 2500000 number of cells, approximately, while the dense mesh has about 3500000 number of cells. It might seem a big difference but the critical geometries are really well meshed in both cases and only the non-critical parts of the turbofan (this is: the flow inside the fan duct) has a smaller number of cells in the standard mesh.

So, for all these reasons, the Standard Mesh has been selected and we will begin the simulation with it.

3.1.4.1 Coarse mesh

The number of control volumes in the mesh is 40, which is a very small number given the size of the turbofan. It can be seen in 3.3 and 3.4 that this mesh cannot be used to run a simulation.

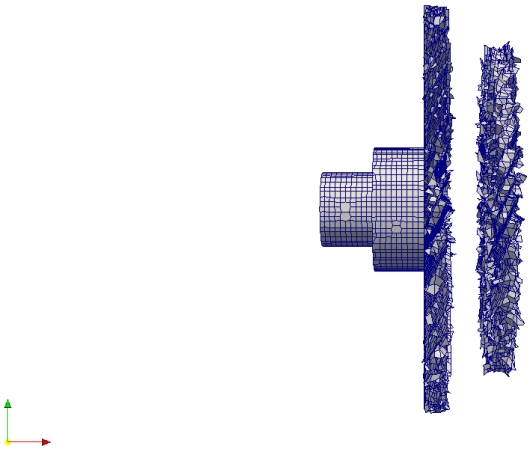

Figure 3.3: Detail of the coarse mesh

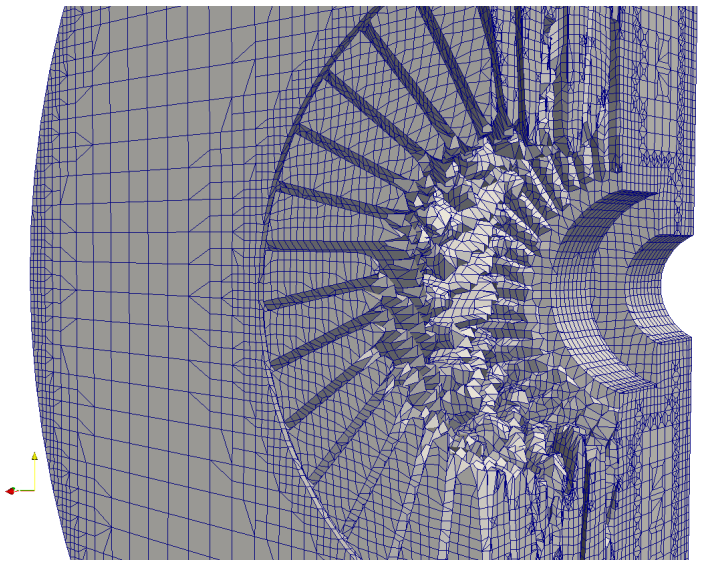

Figure 3.4: Detail of the coarse mesh

3.1.4.2 Standard mesh

Next, it is presented the standard mesh. It has a higher number of cells compared to the coarse mesh, but it cannot be used either given that the size of the turbine is pretty big. The figures 3.5 and 3.6 are a proof of that.

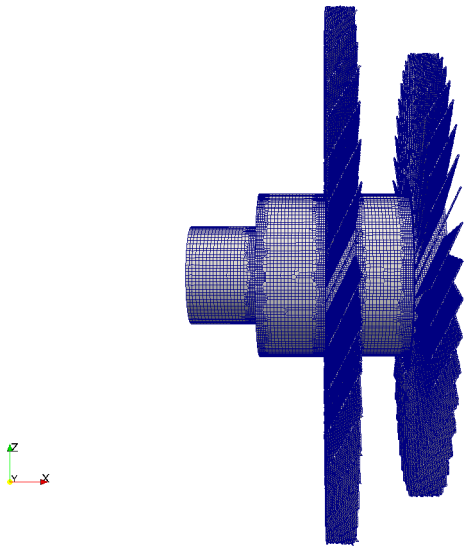

Figure 3.5: Detail of the standard mesh

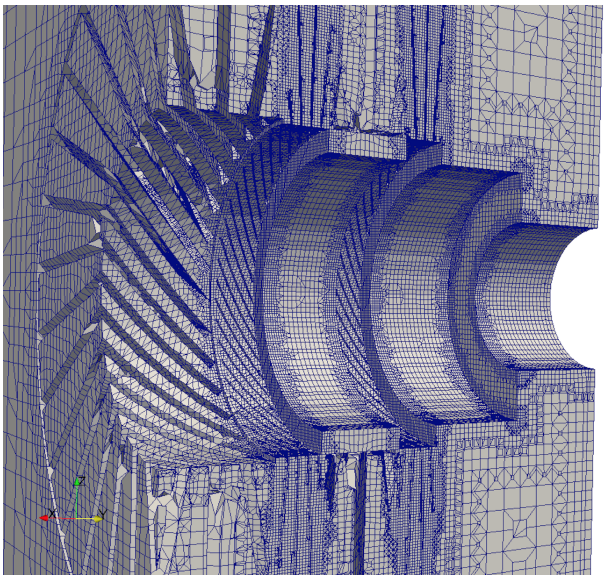

Figure 3.6: Detail of the standard mesh

3.1.4.3 Dense mesh

Finally, the dense mesh is presented in the figures 3.7 and 3.8. This is the best mesh that has been generated for the case but for the reasons commented below, this is not the mesh used to run the simulation.

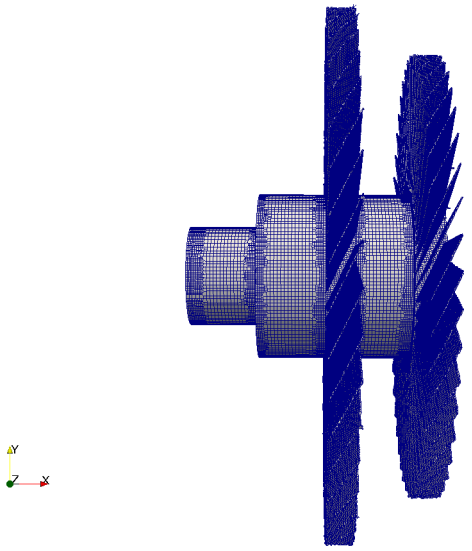

Figure 3.7: Detail of the dense mesh

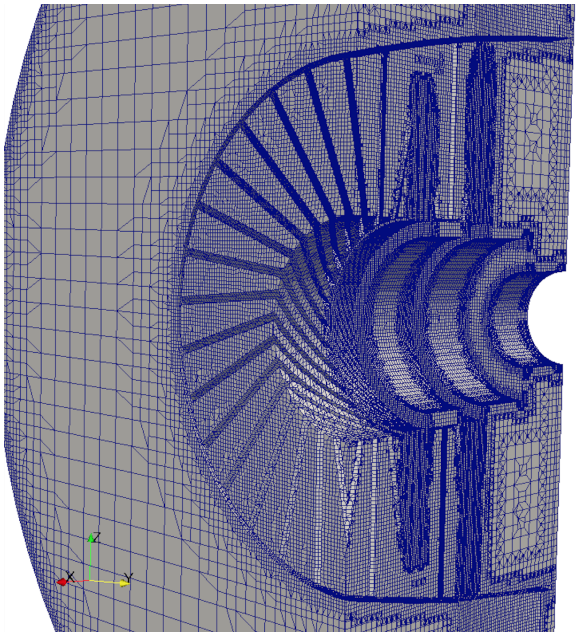

Figure 3.8: Detail of the dense mesh

To end with this section, the *log* file that has been obtained when the mesh has been defined with the *snappyHexMesh* utility is presented below. It has to be taken into account that it is a huge file since the number of iterations is really big and only the final part of the *log* file is shown.

```
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.0                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build  : 4.0-665f1db4c1f1
Exec   : /opt/openfoam4/platforms/linux64GccDPInt32Opt/bin/snappyHexMesh -overwrite
Date   : Dec 05 2016
Time   : 14:17:06
Host   : "victorPC"
PID    : 4504
Case   : /home/victor/Desktop/Final/pimpleDyMFoam/turbofan_std
nProcs : 1
sigFpe : Enabling floating point exception trapping (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using timeStampMaster
allowSystemOperations : Allowing user-supplied system call operations

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


[...]

Snapped mesh : cells:3511624  faces:11081993  points:4202883
Cells per refinement level:
    0 4602
    1 38356
    2 198997
    3 1304463
    4 1965206
Writing mesh to time constant
Wrote mesh in = 153.87 s.
Mesh snapped in = 1259.68 s.
Checking final mesh ...
Checking faces in error :
```

```
    non-orthogonality > 65  degrees                          : 0
    faces with face pyramid volume < 1e-13                   : 0
    faces with face-decomposition tet quality < 1e-15        : 0
    faces with concavity > 80  degrees                       : 0
    faces with skewness > 4   (internal) or 20  (boundary)   : 0
    faces with interpolation weights (0..1)  < 0.02          : 0
    faces with volume ratio of neighbour cells < 0.01        : 0
    faces with face twist < 0.02                             : 0
    faces on cells with determinant < 0.001                  : 0
Finished meshing without any errors
Finished meshing in = 2418.34 s.
End
```

3.2 checkMesh

To view whether the mesh has been generated with error or not, the command `checkMesh` has been typed. This will show some information about the mesh, including the number of faces or cells, as well as some other information. The log obtained is presented below.

```
/*---------------------------------------------------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.0                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
Build  : 4.0-665f1db4c1f1
Exec   : checkMesh
Date   : Dec 05 2016
Time   : 19:24:31
Host   : "victorPC"
PID    : 4598
Case   : /home/victor/Desktop/turbofan
nProcs : 1
sigFpe : Enabling floating point exception trapping (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using timeStampMaster
allowSystemOperations : Allowing user-supplied system call operations

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

--> FOAM Warning :
    From function static Foam::instantList Foam::timeSelector::select0(Foam::Time&, const Foam
    in file db/Time/timeSelector.C at line 270
    No time specified or available, selecting 'constant'
Create polyMesh for time = constant

Time = constant

Mesh stats
    points:           3041731
    faces:            7969935
    internal faces:   7230304
```

```
    cells:            2522441
    faces per cell:   6.026
    boundary patches: 6
    point zones:      0
    face zones:       1
    cell zones:       1

Overall number of cells of each type:
    hexahedra:     2085085
    prisms:        105950
    wedges:        0
    pyramids:      0
    tet wedges:    1267
    tetrahedra:    27
    polyhedra:     330112
    Breakdown of polyhedra by number of faces:
        faces   number of cells
            4   116290
            5   77755
            6   36807
            7   2382
            8   2453
            9   51174
           10   607
           11   770
           12   22688
           13   274
           14   389
           15   15309
           16   30
           17   72
           18   3111
           21   1

Checking topology...
    Boundary definition OK.
    Cell to face addressing OK.
    Point usage OK.
    Upper triangular ordering OK.
    Face vertices OK.
    Number of regions: 1 (OK).
```

```
Checking patch topology for multiply connected surfaces...
                    Patch    Faces   Points                   Surface topology
                    inlet    11685    12488  ok (non-closed singly connected)
                   outlet     6410     7707  ok (non-closed singly connected)
                  LPSpool   537648   631640  ok (non-closed singly connected)
            NacelleStator   172722   217913  multiply connected (shared edge)
                     AMI1     5583     5775  ok (non-closed singly connected)
                     AMI2     5583     5811  ok (non-closed singly connected)
  <<Writing 2 conflicting points to set nonManifoldPoints

Checking geometry...
    Overall domain bounding box (0.77 0.115495 0.106656) (1.35 2.21443 2.20599)
    Mesh has 3 geometric (non-empty/wedge) directions (1 1 1)
    Mesh has 3 solution (non-empty) directions (1 1 1)
    Boundary openness (4.12283e-16 -1.46559e-16 1.42131e-16) OK.
    Max cell openness = 4.10842e-16 OK.
    Max aspect ratio = 10.7936 OK.
    Minimum face area = 8.56533e-10. Maximum face area = 0.002683.  Face area magnitudes OK.
    Min volume = 1.88435e-09. Max volume = 0.000135167.  Total volume = 1.87506.  Cell volumes
    Mesh non-orthogonality Max: 64.9921 average: 11.5854
    Non-orthogonality check OK.
    Face pyramids OK.
 ***Max skewness = 13.9612, 292 highly skew faces detected which may impair the quality of the
  <<Writing 292 skew faces to set skewFaces
    Coupled point location match (average 0) OK.

End
```

3.3 Rotation

An important consideration for this project is that the rotor is spinning at a high speed and this condition has to be somehow communicated to *OpenFOAM*. To work with this condition there are three possible aproaches, which are widely described in [1], the Single Rotation Frame (SRF), the Multiple Reference Frame (MRF) and the Arbitrary Mesh Interface (AMI). In this particular case AMI is going to be used. The method itself is based on defining a sliding mesh where the part of the mesh that moves is rotating in every timestep and the values of the cells lying on the interference are interpolated to update the mesh in every timestep.

To apply this method, a solid containing the "moving" domain has to be defined as a `cellZone`. How to do that has been shown in the previous section. In our case the moving domain is the one contained between the low-pressure compressor and the nearest wall as shown in the figure 3.9.

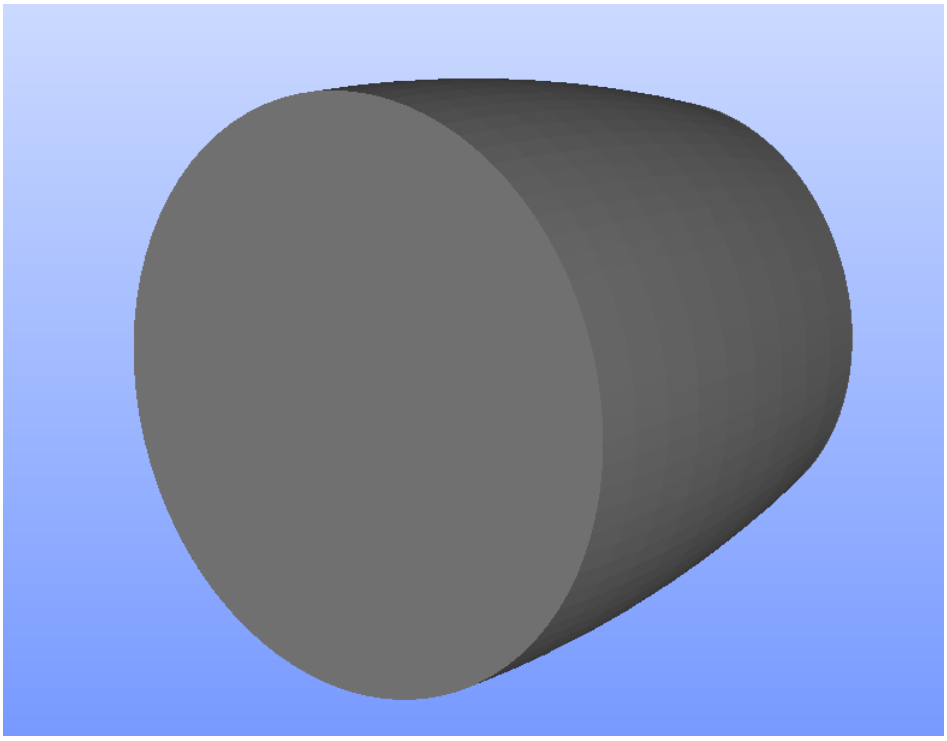

Figure 3.9: Rotating zone solid

To define the parameters related to the rotating zone such as the origin, the axis of rotation, the angular velocity and of course the zone which will be moving the *constant/dynamicMeshDict* file has to be modified with the appropriate parameters for the case. Since the axis of *OpenFOAM* are based on the right hand rule, the rotation axis must be defined as follows: `(1 0 0)`. From information found in different references, it has been chosen 5000 *rpm* as the angular velocity of the rotor. It should be noted that the units used must be those of the International System, so the 5000 *rpm* must be converted to rad/s. Furthermore, an origin point has to be indicated in the axis of rotation; in this case, the origin point is as follows: `(0 1.1675354 1.15542633)`. Finally the file will look like:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      dynamicMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dynamicFvMesh   solidBodyMotionFvMesh;

motionSolverLibs ( "libfvMotionSolvers.so" );

solidBodyMotionFvMeshCoeffs
{
    cellZone        innerAMI;

    solidBodyMotionFunction  rotatingMotion;
    rotatingMotionCoeffs
```

```
    {
        origin      (0 1.1675354 1.15542633);
        axis        (1 0 0);
        omega       523; // rad/s
    }
}


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

Where `cellZone innerAMI;` is used to define the rotating zone. It can clearly be seen that the modification of any parameter is pretty straigh-forward in the file since there is no possible confusion with the names. Another important thing to do when working with AMI is to define 2 patches related to this *"moving zone"*. These patches can be created using the *createPatch* routine. To do so, a dictionary (*createPatchDict*) has to be included in the system folder. For this case, the dictionary file can be as follows:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.3.1                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      createPatchDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

// Do a synchronisation of coupled points after creation of any patches.
// Note: this does not work with points that are on multiple coupled patches
//       with transformations (i.e. cyclics).
pointSync false;
```

```
// Patches to create.
patches
(
    {
        //- Master side patch
        name            AMI1;
        patchInfo
        {
            type            cyclicAMI;
            matchTolerance  0.0001;
            neighbourPatch  AMI2;
            transform       noOrdering;
        }
        constructFrom patches;
        patches (AMI);
    }

    {
        //- Slave side patch
        name            AMI2;
        patchInfo
        {
            type            cyclicAMI;
            matchTolerance  0.0001;
            neighbourPatch  AMI1;
            transform       noOrdering;
        }
        constructFrom patches;
        patches (AMI_slave);
    }
);

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

This method might seem more complicated than simply studying the case with MRF but the results obtained for this particular case are better with the scheme used in this report.

3.4 Boundary conditions

3.4.1 Velocity

Some initial conditions and values have to be defined in order to run the simulation. The initial velocity field and pressure are especially important since these inputs are the starting values of the simulation. So, it is clear that the values at the *inlet*, which was defined within the previous sections, have to be defined. Additionally, the values at the *outlet* of the geometry have to be defined as well, but they can be, unlike the *inlet*, not fixed values. Thus, the following modifications have to be made in the `0.orig/U`.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.0                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      binary;
    class       volVectorField;
    location    "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform (0 0 0);

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform (200 0 0);
    }
```

```
    outlet
    {
        type            inletOutlet;
        inletValue      uniform (0 0 0);
        value           uniform (0 0 0);
    }
    LPSpool
    {
        type            movingWallVelocity;
        value           uniform (0 0 0);
    }
    NacelleStator
    {
        type            noSlip;
    }
    AMI1
    {
        type            cyclicAMI;
        value           uniform (0 0 0);
    }
    AMI2
    {
        type            cyclicAMI;
        value           uniform (0 0 0);
    }
}


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

As it can be seen above, the value of the velocity at the inlet has been set to 200 m/s (about 720 km/h). The *inletOutlet* condition has been set to the outlet (this is: the flow is restricted to move always forwards and reverse flow cannot appear at this region). The NacelleStator file has the noSlip condition given that the fluid is supposed to be Newtonian. Finally, the initial velocity field for the rotational parts AMI1 and AMI2 have been set to 0 (this is: the flow is not moving at the beginning of the simulation).

3.4.2 Pressure

The initial conditions for the pressure have to be evaluated as well in the `0.orig/p` file.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.0                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      binary;
    class       volScalarField;
    location    "0";
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

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

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type            zeroGradient;
    }
    outlet
    {
        type            fixedValue;
        value           uniform 0;
    }
    LPSpool
    {
        type            zeroGradient;
    }
```

```
    NacelleStator
    {
        type            zeroGradient;
    }
    AMI1
    {
        type            cyclicAMI;
        value           uniform 0;
    }
    AMI2
    {
        type            cyclicAMI;
        value           uniform 0;
    }
}


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

The *zeroGradient* condition simply extrapolates the quantity to the current surface from the nearest cell value meaning that the gradient of the quantity is equal to zero in the direction perpendicular to the partch. This condition will not fix a value for the pressure, it will just let the pressure be the same as the calculated using the Navier Stokes equations (provided that the velocity field is known).

Finally, the boundary conditions for the *AMI1* and *AMI2* should not be modified since they are specifically set for a geometry that is rotating.

3.5 Properties of the flow

The properties of the flow have to be defined taking into account the hypotheses made for the case. As mentioned in previous sections, the flow is assumed to be Newtonian. This property can be modified in the `constant/transportProperties` file. Also, since the flow we are dealing with is air, the kinematic viscosity has to be set to $1e^6 m^2/s$.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

transportModel  Newtonian;

nu              [0 2 -1 0 0 0 0] 1e-6;

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

Another hypotheses that has been made in order to alleviate the computation time is that the flow is laminar, although it will not be a very realistic flow. To impose this condition, a modification has to be made in `constant/turbulenceProperties` file and it is presented below.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
```

```
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType  laminar;


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

Since we are only interested in the topology of the flow of this file, other parameters have not been modified.

3.6 controlDict - parameters for the simulation

This file is really important for the simulation since it contains the timesteps and other variables such as the `endTime` or `startTime`. To begin, the initial and final times of the simulation have to be defined. In this case, the `startTime` is 0 s and the `endTime` has been set to $1e^{-3}$s. These changes can be made in the `system/controlDict` file. Furthermore, this file also states the application chosen (*pimpleDyMFoam*), a modified pimpleFoam that handles dynamic meshes.

```
application     pimpleDyMFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         1e-3;
```

The next thing that has to be done is to define the time step for the simulation as well as the write interval (this is; the number of time that must pass in order to make a folder for the simulation in the current simulation time).

```
deltaT .5e-4;
writeControl    timeStep;
writeInterval 1;

purgeWrite      0;

writeFormat     binary;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;
```

The *runTimeModifiable* modifies the deltaT in each iteration to make possible for the solver to converge. Finally, the file that has been used to run the simulation is presented below.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     pimpleDyMFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         1e-4;

deltaT .5e-4;
writeControl    timeStep;
writeInterval 1;

purgeWrite      0;

writeFormat     binary;

writePrecision  6;
```

```
writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

adjustTimeStep  yes;

maxCo           2;

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

Notice that the Courant number has been set to 2.

3.7 fvSchemes and fvSolution

The fvSchemes dictionary contains the numerical schemes that they are used in the simulation. In the hypotheses section we have considered that the flow is transient To impose this condition we must modify the `system/fvSchemes` file, the section of *ddtSchemes*, which is the temporal integration scheme. We must write Euler. It is important to remark that the fle has been modified form the presented in the tutorial because on our case, any turbulent model has been considered so all the variables related to the turbulence had been deleted. The most imprtant change had been done to the *divSchemes* section where we changed the Gauss Linear Upwind scheme to a lower order (and more stable) Gauss Upwind scheme. Finally, the file that has been used to run the simulation is presented below.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
```

```
    grad(p)         Gauss linear;
    grad(U)         cellLimited Gauss linear 1;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss upwind;
    //div(phi,U)      Gauss linearUpwind grad(U);
    div((nuEff*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear limited corrected 0.33;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         limited corrected 0.33;
}


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

The `system/fvSolution` file contains the resolution methods and the tolerances for each equation, in our case, the relaxation factors have been odified to achieve a more stable convergence. Also, the number of PIMPLE iterations have been modified. The fvSolution file including all the changes done is the following:

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
```

```
|  \\    /   O peration     | Version:  4.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                    |
|    \\/     M anipulation  |                                               |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    pcorr
    {
        solver          GAMG;
        tolerance       1e-2;
        relTol          0;
        smoother        DICGaussSeidel;
        cacheAgglomeration no;
        maxIter         50;
    }

    p
    {
        $pcorr;
        tolerance       1e-5;
        relTol          0.01;
    }

    pFinal
    {
        $p;
        tolerance       1e-6;
        relTol          0;
    }

    U
    {
```

```
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-6;
        relTol          0.1;
    }

    UFinal
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-6;
        relTol          0;
    }
}

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

relaxationFactors
{
    "(p|U).*"       0.5;
    UFinal  1;
}

cache
{
    grad(U);
}

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

4 Simulation of the turbofan

4.1 Selection of the solver

The solver used is the *pimpleDyMFoam*. It solves large timestep transient, incompressible flow problems including dynamic meshes using the PIMPLE algorithm which is based on the combination of *SIMPLE* and *PISO* solvers. To correctly understand how the solver works, a briefly explanation of the two solvers will be done in the following pages.

4.1.1 SIMPLE algorithm

The SIMPLE (Semi-Implicit Method for Pressure-Linked Equations) algorithm uses the Navier-Stokes equations for a single-phase flow with a constant density and viscosity written below:

$$\nabla \vec{u} = 0 \tag{4.1}$$

$$\frac{\partial U}{\partial t} + \nabla \Delta(\vec{u}\vec{u}) - \nabla \Delta(\nu \nabla \vec{u}) = -\nabla p \tag{4.2}$$

The pressure equation is deduced from the momentum equation (the procedure can be reviewed in the *OpenFoam* literature) and is:

$$\nabla \left(\frac{1}{a_p} \nabla p \right) = \nabla \left(\frac{H(\vec{U})}{a_p} \right) = \sum_f \vec{S} \left(\frac{H(\vec{U})}{a_p} \right)_f \tag{4.3}$$

The SIMPLE algorithm is used for solving steady-state problems iteratively. In this situations, is not necessary to solve comletely the coupling between velocity and pressure, so the simple algorithm works as follows:

1. Solving the momentum equation gives an approximation of the velocity using as the pressure gradient the previous iteration pressure distribution.

2. The current timestep pressure is later calculated using the pressure equation.

3. The velocities are corrected using the new pressure gradient.

As it can be seen, it uses an implicit method for solving the momentum equation and the pressure equation, but then, it corrects If a steady-state problem is being solved iteratively, it is not necessary to fully resolve the linear pressure-velocity coupling, as the changes between consecutive solutions are no longer small. The SIMPLE algorithm:

4.1.2 PISO algorithm

The PISO (Pressure Implicit with Split Operator) algorithm is an lgoritm used for non-iterative, large time-steps and unsteady problems. It involves one predictor step and two corrector steps an it is designed to fulfill the mass conservation equation. The algorithm is the following.

1. Solve a discetized momentum equation to compute an intermediate velocity field.
2. Compute the mass fluxes at the cell faces.
3. Solve the pressure equation.
4. Correct the mass fluxes.
5. Correct the velocities using the new pressure distribution.
6. Recalculate the boundary conditions.
7. Repeat the process a prescribed number of times.
8. Increase the time step and repeat from 1.

As it can be seenthe predictor step is set when computing the p distribution and then the two corrector steps are applied. With PISO the timestep can be larger than the used with SIMPLE.

In summary, the PIMPLE algorithm is a combinaton between these two schemes, acquiring the capacity to use larger timesteps than SIMPLE and speeding up the simulation since it can use PISO (non iterative) for some of the cells.

4.2 Commands to run the simulation

The first thing to perform the simulation is to copy the tutorial folder case into our current directory. To do so, open a terminal window and type the following command:

```
cp -r /opt/openfoam4/tutorials/incompressible/
      pimpleDyMFoam/mixerVesselAMI2D/* .
```

Then we have to modify all the parameters explained during this report. The first thing that must be done when the *blockMeshDict* has been modified is to type the following lines in a terminal. As it can be seen, it will save the log on a new file named blockMesh.log.

```
blockMesh > blockMesh.log
```

When the *blockMesh* has been generated, the next thing to do is the refinement of the mesh. So, the *snappyHexMesh* utility will be used, provided that all the parameters in `system/snappyHexMeshDict` have been modified. In the terminal, the following command has to be typed:

```
foamJob decomposePar
foamJob -p surfaceFeatureExtract
foamJob -s -p snappyHexMesh -overwrite
```

The previous command will make the mesh refinement to run in parallel (notice the *decomposePar* and the *-p*). Additionally, a log will be created when running the command that will contain all the information about the mesh refinement. The case can be runned without using parallel computing by typing:

```
surfaceFeatureExtract > surfaceFeatureExtract
foamJob -s snappyHexMesh -overwrite
mv log snappyHexMesh.log
```

In order to run the desired simulation, a patch with the AMI surface has to be made. Thus, the following command has to be typed. It will also create a *log* file named *createPatch.log*.

```
createPatch -overwrite > createPatch.log
```

Next, the simulation can be run.

```
cp -r 0.orig 0
foamJob -s PimpleDyMFoam
mv log PimpleDyMFoam.log
```

And finally, if we were working in parallel (in that case a -p must be added to the pimpleDyMFoam command as we did in the previous steps), the case has to be reconstructed to show the results.

```
reconstructParMesh -constant
reconstructPar
```

There has also been added into the folder the `Allrun` file, which will run the case automatically.

To view the results, typing `paraFoam` in the `terminal` window should be enough. The .zip folder of the case delivered along with this report contains an Allrun and an Allclean scripts to automatize all these commands.

5 Post-process

5.1 Analysis of the results

In this section, there are going to be presented several graphs with the different parameters that have been obtained in the simulation of the case. The following plots are the results of all the commands that have been mentioned in the previous sections and are divided into two sections

5.1.1 Velocity

5.1.1.1 Overall look

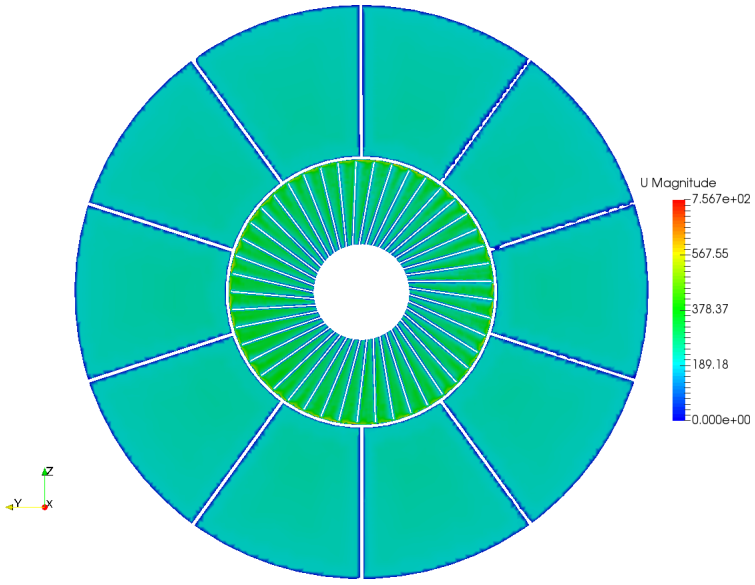

Figure 5.1: Total speed

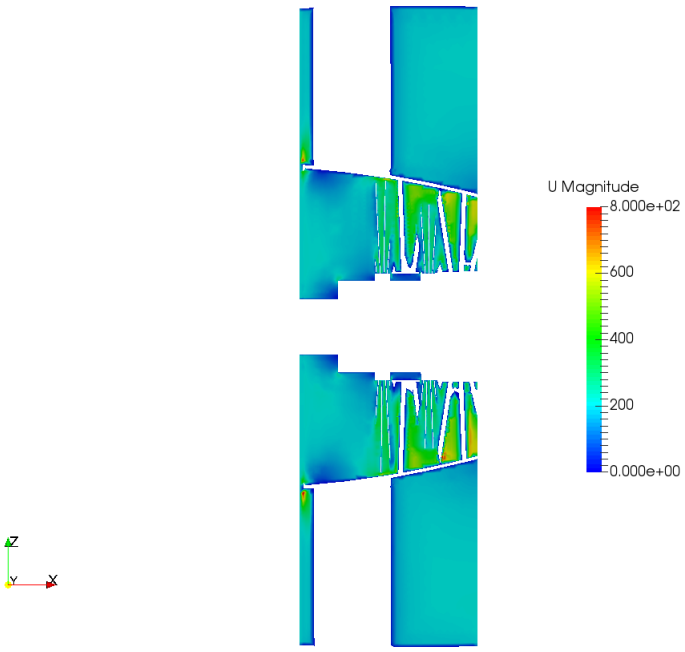

Figure 5.2: Total speed

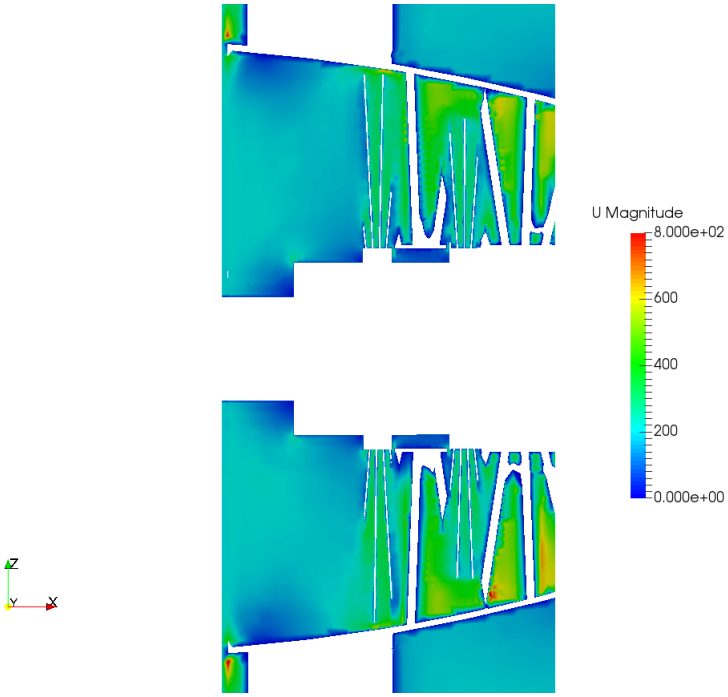

Figure 5.3: Detail of the total speed

5.1.1.2 Velocity in the x direction

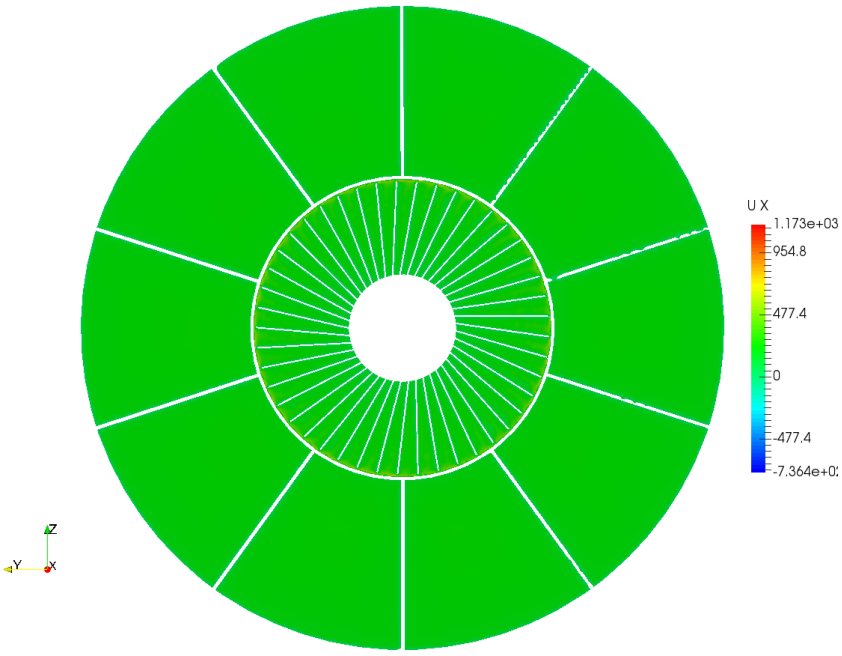

Figure 5.4: Speed in the x direction

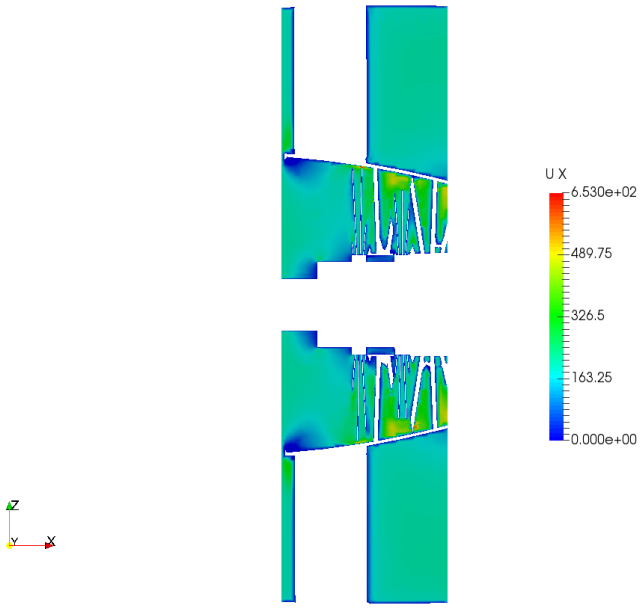

Figure 5.5: Speed in the x direction

5.1.1.3 Velocity in the y direction

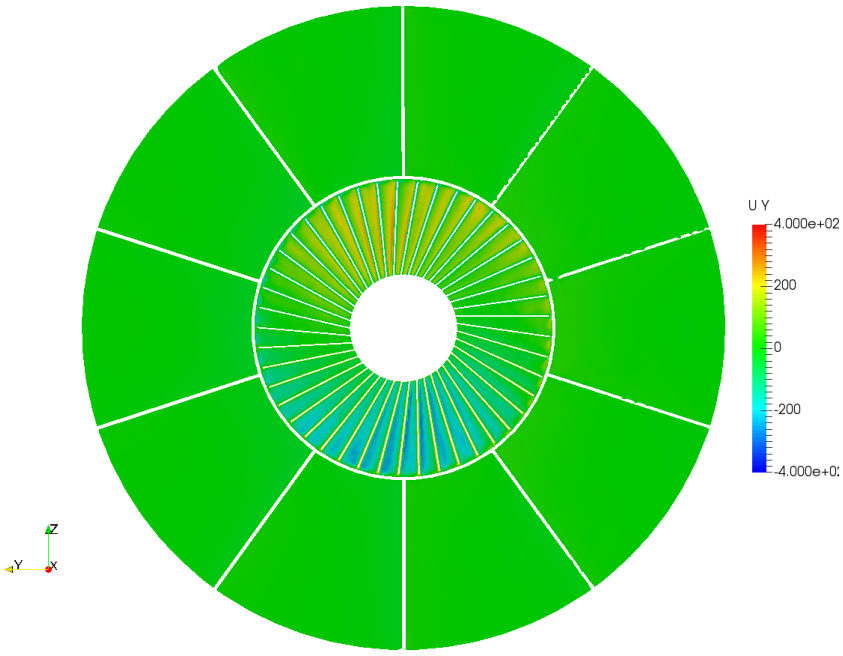

Figure 5.6: Speed in the y direction

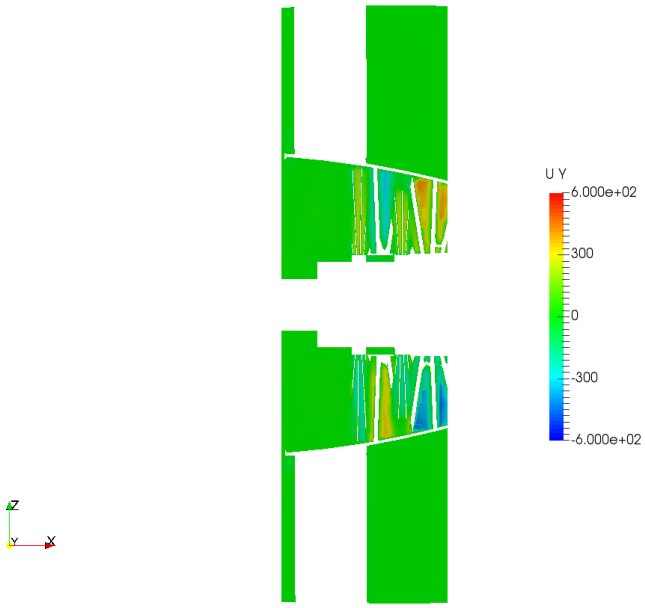

Figure 5.7: Speed in the y direction

5.1.1.4 Velocity in the z direction

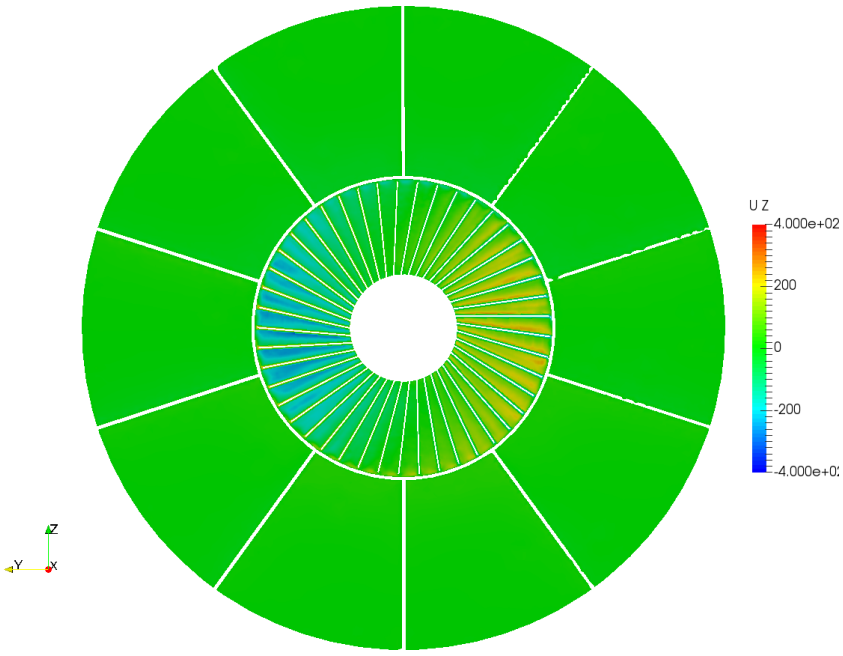

Figure 5.8: Speed in the z direction

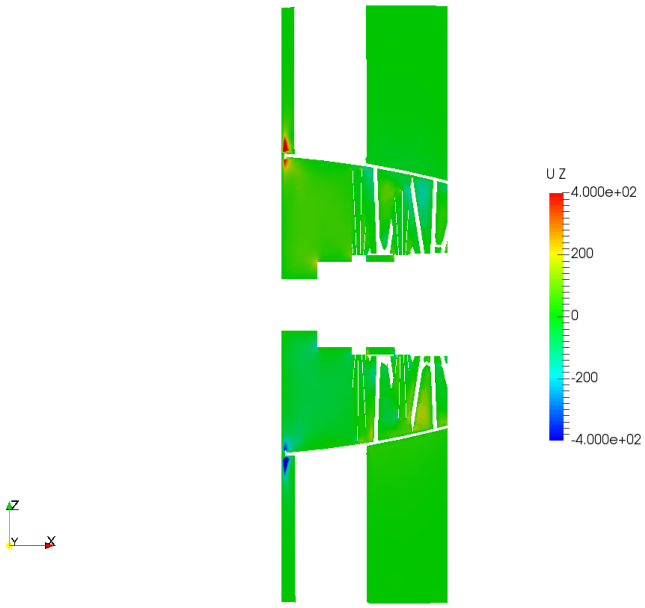

Figure 5.9: Speed in the z direction

5.1.2 Pressure

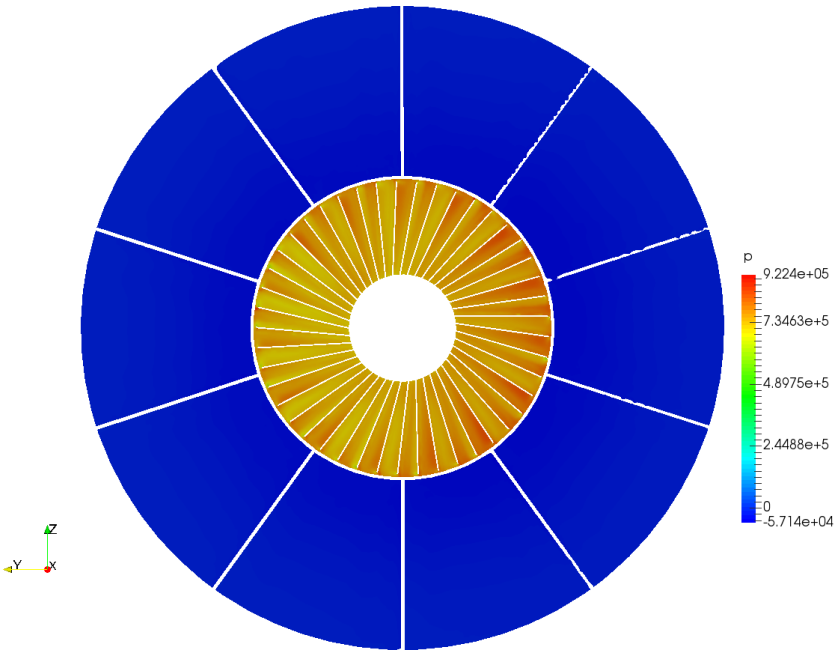

Figure 5.10: Pressure

Figure 5.11: Pressure

5.1.3 Comments

5.2 Conclusions and Further work

The objective of this course and, in particular, of this project is to get in touch with open-source CFD software and the other open-source applications related to it such as Salome.

We believe that this goal has been more than fullfilled by choosing a case as complex as the one discussed in this report whic includes things not viewed in the theory sessions such as the rotating meshes. In addition, as a side effect, we have also learned how to handle a operating system as Linux.

6 Difficulties that we faced

We have faced with several difficulties during the simulation of this case. First of all, our computers do not have enough processing power to simulate a complicated geometry and some hypotheses have been made in order to facilitate the simulation. Furthermore, with the bootable-USB we could not create a dense mesh and run the application; therefore, we had to make a partition on the computer.

In relation to the mesh, we had several problems because some of the downloaded *GrabCAD* files presented construction errors and some parts of the low pressure compression stage studied were *disjointed*. To solve this problem, a modification of the files in *SolidWorks* was necessary. It has to be mentioned that this was a very tedious process since all these geometries had to be joined manually.

Another important aspect to highlight is that the flow that goes through a turbofan is highly compressible but we have assumed that it was an incompressible flow. The solvers used to simulate compressible flow are really difficult to work with and we could not find a solution to deal with it. So, when we faced this situation, we decided that the best way to proceed with the case was to use an incompressible solver (*pimpleDyMFoam*).

The flow will not be very realistic with all the hypothesis made; however, it does show how the flow behaves when it faces with the low pressure compression stage of a turbofan and the results, although not real, give a very good idea of what happens to the flow in terms of the velocity field.

7 References

[1] Fumiya Nozaki. CFD for Rotating Machinery. (May), 9240.

[2] Jordi Casacuberta. Study of fluid dynamics applications in the field of engineering by using OpenFOAM. 2014.

[3] Learner Feedback. Running in parallel. pages 45–54.

[4] Eric Paterson. PENN STATE The icoFoam " Cavity Tutorial ". (April), 2010.

[5] Olle Penttinen. A pimpleFoam tutorial for channel flow, with respect to different LES models. *Change*, pages 1–23, 2011.

[6] Olivier Petit. Mesh generation: different ways of creating the mesh. 2011.

[7] Roberto Pieri. A tool for pre-processing : snappyHexMesh. (June), 2014.

[8] Jone Rivrud Rygg. CFD Analysis of a Pelton Turbine in OpenFOAM. *Thesis*, (June):119, 2013.