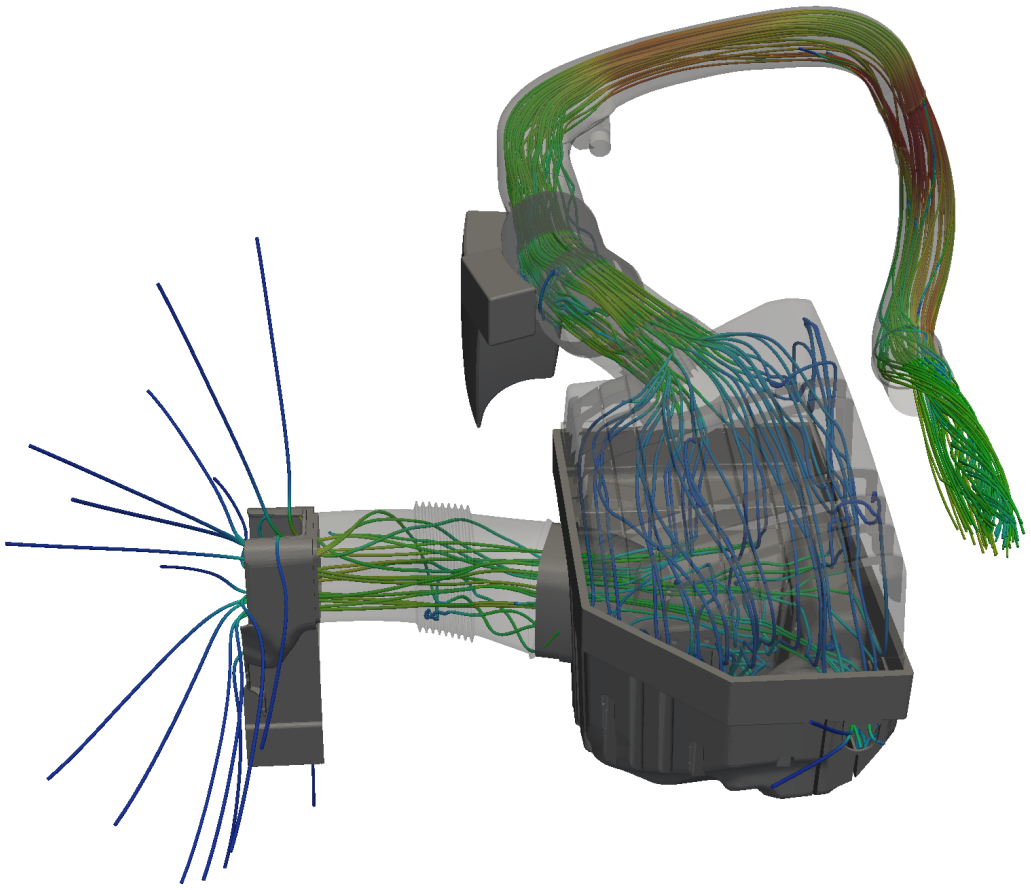
CHALMERS



CFD for air induction systems with OpenFOAM

Master's thesis in Fluid Mechanics

KLAS FRIDOLIN

Department of Applied Mechanics
Division of Fluid Mechanics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2012
Master's thesis 2012:13

MASTER'S THESIS IN FLUID MECHANICS

CFD for air induction systems with OpenFOAM

KLAS FRIDOLIN

Department of Applied Mechanics

Division of Fluid Mechanics

CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2012

CFD for air induction systems with OpenFOAM
KLAS FRIDOLIN

Master's thesis 2012:13
ISSN 1652-8557
Department of Applied Mechanics
Division of Fluid Mechanics
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
Air induction system with streamlines coloured by velocity magnitude

Chalmers Reproservice
Gothenburg, Sweden 2012

CFD for air induction systems with OpenFOAM
Master's thesis in Fluid Mechanics
KLAS FRIDOLIN
Department of Applied Mechanics
Division of Fluid Mechanics
Chalmers University of Technology

Abstract

CFD is a very important tool in today's design of automotive air induction systems. The simulations enable the flow in a proposed design to be evaluated without manufacturing the system. Most commercial software's used for this are expensive, and in the ever increasing competition lowering costs is a key issue for a carmaker such as Volvo Car Corporation.

This thesis evaluates the use of the open source CFD code OpenFOAM for simulating the flows in air induction systems. Two test cases are used and compared to the commercial code Fluent sold by ANSYS, Inc. The results from both codes are also compared to results from physical tests.

Results from the test cases are that although a lot of time has been spent on finding viable numerical schemes and solver settings, there is still a lot of work to be done before the OpenFOAM simulations show the same numerical stability as the Fluent simulations. Because of this and that the same boundary conditions could not be used OF shows results further from the experimental results than Fluent. During the project a problem in OpenFOAM concerning oscillating velocities in the interface to porous media was discovered. Put together there is still work to be done before OpenFOAM can be used for complete air induction system simulations.

When there is no air filter involved, OpenFOAM is very suitable to use in topology optimisation. Since it is command line based it is easy to couple with optimisation software such as modeFrontier. Also investigated in this thesis is the adjoint solver in OpenFOAM. It is a simple adjoint optimiser but to be truly usable it needs extension. Adjoint optimisation in general is very useful as a guide to designers, but the results are not suitable to use directly since it doesn't take variables such as manufacturability into account.

Keywords: CFD, OpenFOAM, Air induction system, porous media, adjoint, topology optimisation

Sammanfattning

CFD är idag ett viktigt verktyg vid konstruktion av luftintagssystem till fordon. Simuleringar möjliggör utvärdering av konstruktioner utan att tillverka ett fysiskt system. De flesta kommersiella programvaror som används idag är dyra, och med den ökande konkurrensen inom bilindustrin är låga kostnader mycket viktigt.

Det här arbetet har utvärderat om den öppen källkods baserade CFD koden OpenFOAM skulle kunna användas för flödessimuleringar i luftintagssystemen. Två testsystem har simulerats och jämförts såväl med den kommersiella programvaran Fluent från ANSYS, Inc. som med fysiska tester.

Resultaten från testfallen visar att trots att mycket tid lagts på att få OpenFOAMsimuleringarna numeriskt stabila är de ännu inte på samma nivå som simuleringarna där Fluent har använts. Detta tillsammans med att samma randvillkor inte kunde användas gör att resultaten från OF ligger längre ifrån de experimentella mätningarna än de som erhållits med Fluent. Ett problem med OpenFOAM har också upptäckts under projektets gång. I gränsskiktet till poröst media (luftfilter) förekommer icke fysikaliska hastighetsoscillationer. Ovanstående sammantaget så krävs det fortfarande en del arbete innan OpenFOAM kan användas som ersättare till Fluent för kompletta luftfiltersystemsimuleringar. OpenFOAM är dock mycket lämpligt att använda vid topologioptimering när luftfiltret inte behöver inkluderas i simuleringen. Eftersom det är kommandoradsbaserat är det mycket lätt att koppla samman med optimeringsprogram såsom modeFrontier. I projektet har också adjointlösaren distribuerad med OpenFOAM undersökts. Det är en enkel men användbar lösare men för att vara riktigt användbar behöver den byggas ut. Generellt gäller för optimering med adjointmetoden att resultaten är användbara som en guide till konstruktörer men att resultaten inte går att använda rakt av då faktorer som tillverkningsbarhet ej tas i beaktande.

Acknowledgements

First I would like to thank my supervisor Henrik Nyberg at VCC for letting me into the Volvo and open-source CFD worlds. Both very interesting and challenging meaning the learning curve has been very steep. This makes me thankful for always answering sometimes novice questions.

Many thanks also goes out to Johan Brunberg and Oscar Bergman, CAE analysts at the air induction system group for never ending patience when my knowledge hasn't sufficed. Also important for the project has been the very helpful nature of Urban Engberg from the VCC CAE support team whom has provided help regarding the computational facilities.

To the rest of the people in the air induction system group, thanks for treating me as one of you.

Last but not least, huge thanks to associate professor Håkan Nilsson at Chalmers University of Technology for acting as examiner and answering questions.

Nomenclature

Abbreviations

Abbreviation	Description
AIS	Air Induction System
AMM	Air Mass Meter
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CFD	Computational Fluid Dynamics
MRF	Multiple Reference Frames
RANS	Reynolds Averaged Navier-Stokes
OpenFOAM	Open source Field Operation And Manipulation
UI	Uniformity Index
URF	Under Relaxation Factor
VCC	Volvo Car Corporation

Roman symbols

Symbol	Description	Units
c_p	Specific heat capacity at constant pressure	$[J/(kgK)]$
k	Turbulent kinetic energy	$[m^2/s^2]$
p	Static pressure	$[Pa]$
p_0	Total pressure	$[Pa]$
R	Universal gas constant	$[J/(molK)]$
T	Temperature	$[K]$

Greek symbols

Symbol	Description	Units
γ	Uniformity index	-
δ_{ij}	Kronecker delta	-
η	Kolmogorov length scale	$[m]$
ϵ	Turbulent energy dissipation rate	$[m^2/s^3]$
μ	Dynamic viscosity	$[Pa\ s]$
ν_η	Kolmogorov velocity scale	$[m/s]$
τ_η	Kolmogorov time scale	$[s]$
Ψ	Lagrange multiplier	-

Contents

1 Introduction

1.1 Background

In the beginning of the life of the car, oil was cheap and seemed endless and no one had heard of greenhouse gases or global warming. In recent years though, more and more people have become aware of these things and the price of oil has increased a lot during the past decades. This has driven consumers to request more fuel efficient cars and therefore the automotive industry puts a great deal of effort into reducing the fuel consumption of their engines. Better fuel economy brings several advantages with it such as longer range, a lower fuel bill for the vehicle owner and less pollution. But, probably the greatest advantage is that it makes the petrol/diesel powered car a more sustainable transport solution. One of the many steps in reducing engine fuel consumption is reducing the pressure losses in the air induction system. In the air induction system high airflows have to be reached while keeping the pressure losses at a minimum. The important functions of the air induction system can be described in the following points:

- Transport the air needed for combustion to the engine within specified leakage limits.
- Reduce noise to levels decided by regulations or VCC.
- Clean the air from dust and dirt.
- Prohibit snow and water intrusion.
- Maintain a favourable profile of the flow with a pressure drop within an acceptable level.

The problem is that the ducting of the air induction system has strict packaging constraints to fit under the hood of the car together with all the other systems present in a modern car. Figure 1.1.1 shows an example of the complicated topologies required for the air induction system to fit under the hood.

VCC has a group called AIS which is responsible for the air induction system including the air filter and resonators but not intercoolers in the case of forced induction engines. The group uses CFD simulations to predict the flow in the ducts and to estimate the pressure loss and velocity uniformity before the filter (for maximum filter utilisation), before the air mass meter and at the outlet. Of course, the simulations are verified by physical testing in a flow rig before they reach production status.

In todays CAE work the limiting factor on the amount of simulations possible are the number of licenses for the softwares used. As a solution to this problem VCC wants to examine the possibilities of using the open source software OpenFOAM to carry out some of the CFD simulations. This would lead to both lower costs for software licenses and better utilisation of available hardware resources. As a first step OpenFOAM would likely be used for optimisation runs where lots of software licences are often required to achieve reasonable lead times.

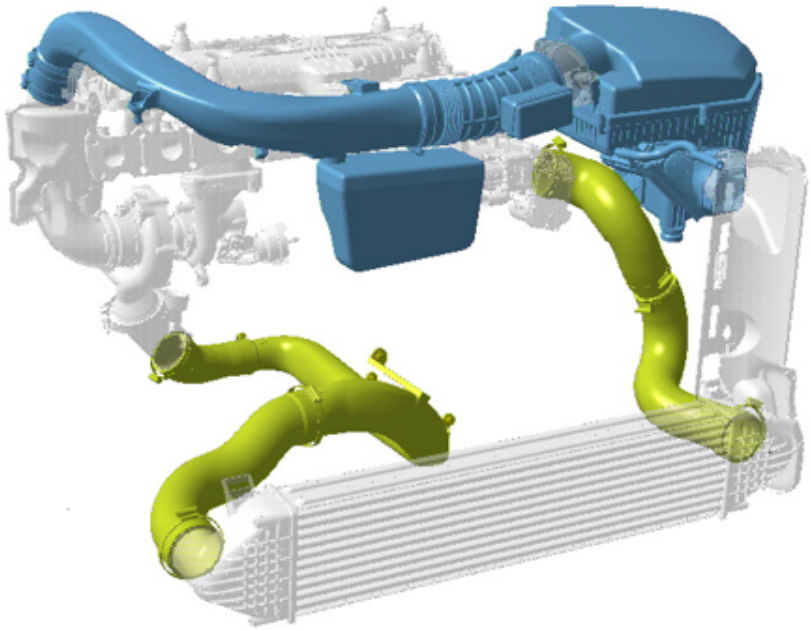


Figure 1.1.1: *Air induction system for the 5 cylinder diesel engine in the Volvo EuCD platform*

1.2 Objectives

The main objective is to assess if OpenFOAM with satisfying results can simulate the flow in air induction systems. The results are to be compared both against existing simulations using commercial software (mainly ANSYS Fluent) and against results from experiments. The OpenFOAM method will involve necessary pre-processing steps, (identification of boundary conditions, mesh conversion/merging etc.), compressible/incompressible and temperature dependant simulations of ducts and filter systems and necessary post-processing steps. Important data from post processing include uniformity index and total pressure losses. It is also of great importance that the developed method is relatively simple to use as it otherwise will be too time-consuming to use in day-to-day work.

If OpenFOAM is deemed a feasible software to use for the air induction system simulations, the next step is the investigation of topology optimisation methods using OpenFOAM. This will require a study of optimisation methods to decide which one to use in conjunction with OpenFOAM. The optimisation would focus on finding the optimal design with regard to minimising pressure losses while at the same time achieving high flow uniformity with given packaging constraints.

1.3 Limitations

The evaluation of OpenFOAM as a computational tool is restricted to the air induction systems only. The thesis will not try to improve existing calculation methods by, for example, using a new turbulence model, but instead try to replicate them using OpenFOAM.

OpenFOAM is open source and it is therefore possible to make changes to the code and change the functionality to suit most needs. This is one of the great advantages with OpenFOAM, but outside of the scope of this thesis. Instead the focus will be on evaluating the performance of existing functionality and as such changes in the code will not be made.

2 Theory

This chapter will present a brief introduction to the theory used in this thesis. Considering the vast size of the field of fluid mechanics, focus will be on aspects relevant for the work carried out.

2.1 Fluid mechanics

A fluid is a substance that is unable to support shear stresses [1], though in reality they can withstand some because of viscosity. The study of fluids is called fluid mechanics, and the part of fluid mechanics used here is called fluid dynamics and studies forces in conjunction with fluids.

2.1.1 Governing equations

Fluid flows are well described by the Navier-Stokes equations (eq. 2.1.1), also called the momentum equations since they describe the relation between momentum, pressure and viscous forces in a fluid.

$$\frac{\partial}{\partial t}\left(\rho v_i\right) + \frac{\partial}{\partial x_j}\left(\rho v_i v_j\right) = -\frac{\partial p}{\partial v_i} + \frac{\partial}{\partial x_j}\left(\mu\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\right) + S_M \tag{2.1.1}$$

The momentum equations (one equation in each coordinate direction) together with the continuity equation (eq. 2.1.2), energy equation (eq. 2.1.3) and an equation of state (the perfect gas law has been used in this thesis, eq. 2.1.4)can be seen as the governing equations of compressible fluid dynamics.

$$\frac{\partial}{\partial t}\left(\rho u_i\right) + \frac{\partial}{\partial x_j} = 0 \tag{2.1.2}$$

$$\frac{\partial(\rho i)}{\partial t} + \frac{\partial}{\partial x_j}\left(\rho i v_j\right) = -p\frac{\partial v_j}{\partial x_j} + \frac{\partial}{\partial x_j}\left(k\frac{\partial T}{\partial x_i}\right) + \Phi + S_i \tag{2.1.3}$$

$$p = \rho R T \tag{2.1.4}$$

If the flow is incompressible and temperature independent only the Navier-Stokes and continuity equation are needed. In this thesis, both incompressible and compressible flows are used, since the flow speeds can be higher than Mach 0.3 in thinner sections.

A flow is either laminar or turbulent, with the Reynolds number being the deciding factor. When the Reynolds number of a flow is above a certain critical value the flow becomes turbulent.

2.1.2 Turbulence

In laminar flow the adjacent layers slide past each other in an orderly fashion. Turbulent flow on the other hand is random and chaotic in its nature. There are six main characteristics for turbulent flow:

- Irregular
- Diffusive
- Has a relatively high Reynolds number
- Three-dimensional
- Dissipative
- Property of the flow (i.e. not the fluid)

In turbulence unsteady vortices called eddies appear. The largest eddies extract energy from the mean flow, called turbulent kinetic energy. This energy is then transferred to smaller and smaller eddies until it reaches the smallest ones where it is dissipated into heat. This process of energy transfer is called the cascade process. The scale of the smallest eddies are called the Kolmogorov scales and where defined in 1941 by A.N. Kolmogorov [2]. There are three Kolmogorov scales: length, velocity and time and they are defined as :

$$\eta = \left(\frac{\epsilon^3}{\nu}\right)^{\frac{1}{4}} \tag{2.1.5}$$

$$\tau_\eta = \left(\frac{\epsilon}{\nu}\right)^{\frac{1}{2}} \tag{2.1.6}$$

$$\upsilon_\eta = (\nu\epsilon)^{\frac{1}{4}} \tag{2.1.7}$$

In theory one could resolve all of the turbulent scales. The problem with this is that the Kolmogorov scales are usually very small which means that this would require a tremendous amount of computing power and memory. Therefore resolving all the turbulent scales is only an option (with todays computers) for simple problems with low Reynolds numbers. To compute more realistic problems some kind of turbulence modelling has to be used.

Reynolds Averaged Navier-Stokes

The Reynolds averaged Navier-Stokes (RANS) equations are time-averaged versions of the original equations. They are obtained by introducing the Reynolds decomposition of the pressure and velocity and then time-averaging the equations. The Reynolds decomposition consists of dividing the instantaneous pressure and velocity into one time-averaged and one fluctuating part, i.e.

$$u = U + u' \tag{2.1.8}$$

$$p = P + p' \tag{2.1.9}$$

After insertion of the decomposition and time-averaging of the Navier-Stokes equations one additional term appears, $-\rho\overline{u_i' u_j'}$. This term is called the Reynolds stress tensor and means additional unknowns have been introduced. Since the number of equations has not increased, this creates a closure problem. To solve this the Reynolds stress tensor can be modelled using an eddy viscosity and the velocity gradients. This is called the Boussinesq assumption (Eq. 2.1.10) and the basic idea behind it is to model the small-scale eddies with a viscosity term.

$$-\rho\overline{u_i' u_j'} = \mu_t \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\frac{\partial u_k}{\partial x_k}\delta_{ij} \right) - \frac{2}{3}k\delta_{ij} \tag{2.1.10}$$

δ_{ij} is the Kronecker delta and k is the turbulent kinetic energy defined as

$$k = \frac{1}{2}\left(u'^2 + v'^2 + w'^2\right) \tag{2.1.11}$$

There are plenty of models to describe the turbulent quantities including one-equation models, two-equation models and algebraic models.

Realizable $k - \epsilon$ model

To compute the turbulent quantities the realizable $k - \epsilon$ model is normally used at VCC, and as such also in this thesis. The realizable $k - \epsilon$ model is a two-equation model by Jones and Launder [3] with one equation for k and one for ϵ. The turbulent dissipation rate ϵ is a small scale variable but it is here used as the dissipation rate for large eddies. This is possible because the dissipation rate of the small eddies matches the rate at which the large eddies extract energy from the mean flow through the energy spectrum [4]. The two additional transport equations for k and ϵ are defined in equation 2.1.12 and 2.1.13 respectively.

$$\frac{\partial}{\partial t}(\rho k) + \frac{\partial}{\partial x_i}(\rho k u_i) = \frac{\partial}{\partial x_j}\left[\left(\mu + \frac{\mu_t}{\sigma_k}\right)\frac{\partial k}{\partial x_j}\right] + P - \rho\epsilon \tag{2.1.12}$$

$$\frac{\partial}{\partial t}(\rho\epsilon) + \frac{\partial}{\partial x_j}(\rho\epsilon u_j) = \frac{\partial}{\partial x_j}\left[\left(\mu + \frac{\mu_t}{\sigma_\epsilon}\right)\frac{\partial \epsilon}{\partial x_j}\right] + \rho C_1 S\epsilon - \rho C_2 \frac{\epsilon^2}{k + \sqrt{\nu\epsilon}} + C_{1\epsilon}\frac{\epsilon}{k} \tag{2.1.13}$$

where

$C_{1\epsilon} = max\left[0.43, \frac{\eta}{\eta+5}\right]$, $\eta = S\frac{k}{\epsilon}$, $S = \sqrt{2S_{ij}S_{ij}}$, $\mu_t = \rho C_\mu \frac{k^2}{\epsilon}$ and $C_\mu = \frac{1}{A_0 + A_s}\frac{kU^*}{\epsilon}$

The model constants as specified in OpenFOAM are the following: $C_\mu = 0.09$, $A_0 = 4$, $C_2 = 1.9$, $\sigma_k = 1$ and $\sigma_\epsilon = 1.2$. For further details of the realizable $k - \epsilon$ model, please see Jones and Launder [3].

3 Methodology

This chapter contains a brief description of the softwares used, a description of how the air filter system simulations were setup and an introduction of the adjoint method.

3.1 Softwares used

3.1.1 OpenFOAM version 2.1.0

OpenFOAM is in this thesis being evaluated as a CFD solver for air induction systems, but it is more than that. The core of OpenFOAM is a C++ toolbox that enables the creation of customised numerical solvers. The distribution also contains a wide selection of solvers designed to solve specific problems. For more information on OpenFOAM and how to set up a case, please see appendix A.

3.1.2 ANSYS Fluent 14.0

The currently used CFD software for flow calculations in the air induction system group is the commercial package Fluent. It contains a variety of physical modelling capabilities to model most kinds of flow situations encountered in engineering. Fluent is the software against which the OpenFOAM calculations in this thesis have been compared.

3.1.3 ANSA version 13.1.5

ANSA is the pre-processing tool of choice for meshing the air induction systems. Models are created in the CAD system CATIA V5 and imported into ANSA for meshing. One of the great advantages of ANSA for this thesis project is that it can export meshes directly into OpenFOAM format. This means that it generates the file structure of an OpenFOAM case directly. It is also possible to set some solver settings in ANSA, although the selection of solvers isn't complete.

3.1.4 ParaView version 3.4.0

This is the main post-processing software distributed with OpenFOAM. It is an open-source data analysis and visualisation tool and it can either be invoked by using paraFoam wrapper script or by converting the data to VTK format and open with ParaView. Data processing can be done either interactively in a 3D environment or using command line batch processing.

3.1.5 CATIA V5

CATIA is a CAD/CAE/CAM system developed by Dassault Systémes. It is widely used in the automotive and aerospace industries and is also the primary CAD system used at VCC. In this thesis it has been used to create the models of the air induction systems simulated.

3.1.6 swak4foam version 6.3.7

Swak4foam stands for "Swiss Army Knife for Foam" and is a set of libraries and utilities to perform various kinds of data manipulation and representation. It can be used to set complicated boundary conditions depending on expressions. In this thesis the part of swak4foam that contains extended *functionObjects* (used to perform calculations after each iteration) has been used to compute total pressure and uniformity index at internal sections.

3.2 Air filter system simulations

The currently performed air filter simulations are run as steady state simulations in several stages. This is because it can be difficult to reach convergence if all of the options such as second order accurate numerical schemes were to be used from the start of the simulation. As stated in section 1.3 this thesis is not about improving existing simulation methods, but they will be used as a starting point when setting up the OpenFOAM simulations. Fluent has been used for a relatively long time at the department and boundary conditions etc. are tweaked to suit this software package, which is why all options can not be directly used with another software package.

3.2.1 Simulation setup

Geometry and mesh generation

The models used are created in the CAD system CATIA V5 and the computational mesh is created in ANSA. The geometry is split up into separate volumes with interior surfaces in between. This is to simplify volume meshing and also to get interior surfaces which can be used for evaluation. Also, the air filter itself must be a separate *cellZone* for the solver to know where to apply porosity settings (see section 3.2.1). The volume mesh consists of tetrahedral elements. To increase accuracy in the boundary layers 2 prism layers of ca 1 mm thickness each are used along the walls of the ducts. A part of the mesh with layers are shown in figure 3.2.1.

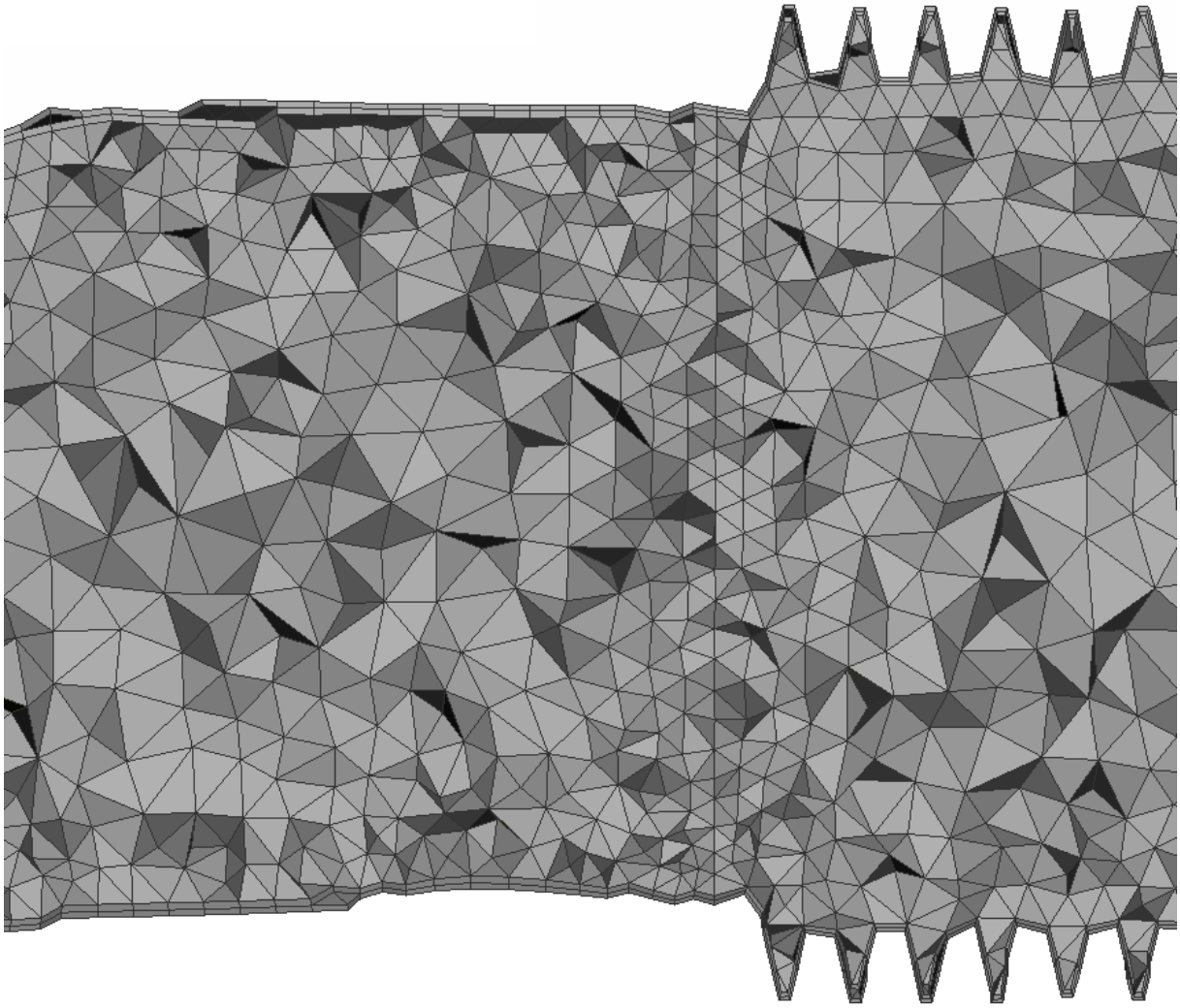


Figure 3.2.1: *Mesh with prism layers along walls*

No prism layers are created in the inlet spheres, the air filter box or in the volume around the AMM (usually located after the air filter box). To take entrance losses of the air intake into account a half sphere is added at every inlet. At the exit of the clean air duct, a volume extrusion is added to reduce the outlet boundary effects. Figure 3.2.2 shows a model with and without the added bulbs and outlet extrusion.

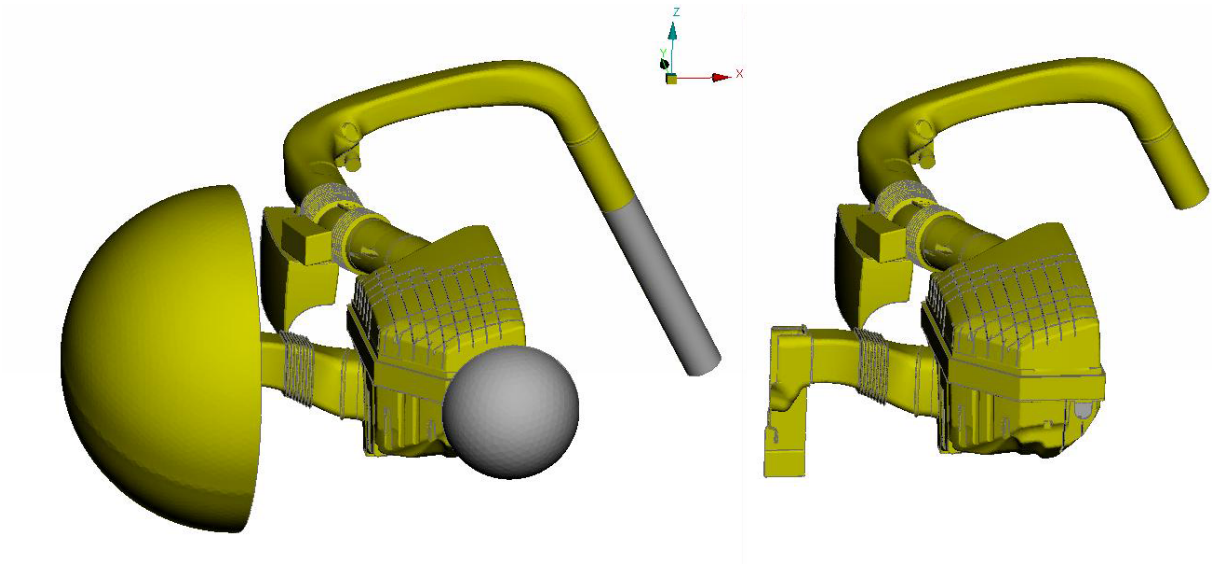


Figure 3.2.2: *Example model with and without added spheres at inlets and extrusion at outlet*

ANSA has native OpenFOAM support and by using this the mesh was directly exported into OpenFOAM format. The version of ANSA used (13.1.5) does not completely support the file structure and syntax used in the current OpenFOAM version (2.1.0). This necessitated some manual editing of solver settings and boundary conditions. An alternative way would have been to export the mesh in Fluent format and then convert it with the OpenFOAM utility *fluentMeshToFoam*. After the mesh was exported to OpenFOAM the utility *checkMesh* was used to check the mesh for skewness and other mesh quality parameters.

Choice of solver

OpenFOAM comes with a long list of different solvers written for different applications. The requirements on the solver for the air filter system simulations are:

- Steady state
- Compressible
- Able to handle porous zones

These requirements made it easy to choose a solver, since only the *rhoPorousMRFSimpleFoam* solver handles all of these demands. It uses the SIMPLE algorithm for pressure-velocity coupling, RANS turbulence modelling (see section 2.1.2) and handles both implicit and explicit porosity treatment. The solver also handles MRF, Multiple Reference Frames, which is used when a part of the mesh needs to move in relation to the rest (not used in this thesis).

Boundary conditions

The OpenFOAM boundary conditions found to work best for the air filter systems are presented in table 3.2.1. The *inletOutlet* boundaries are set to zero to prevent back flow. If the flow is in the forward direction they act as homogenous Neumann boundaries.

Table 3.2.1: OpenFOAM boundary conditions used for the air filter systems

Quantity	Inlet(s)	Outlet	Walls
U	*flowRateInletVelocity*	*inletOutlet*	No-slip
p	*zeroGradient*	*fixedValue*	*zeroGradient*
k	*turbulentIntensityKineticEnergyInlet*	*inletOutlet*	*compressible::kqRWallFunction*
epsilon	*compressible::turbulentMixingLengthDissipationRateInlet*	*inletOutlet*	*compressible::epsilonWallFunction*
T	*fixedValue*	*inletOutlet*	*zeroGradient*
mut	*calculated*	*calculated*	*mutkRoughWallFunction*
alphat	*calculated*	*calculated*	*alphatWallFunction*

Setting the *flowRateInletVelocity* with a negative magnitude at the outlet (resulting in an outlet boundary) has also been tested. This approach would be preferable but produces significantly less stable simulations. Since setting the flow rate at the outlet doesn't produce stable results, cases with several inlets has to have the mass flows for the respective inlets determined. This has been done by using an incompressible solver with which it has been possible to run a simulation with an outlet having a negative magnitude flow rate. This simulation has been run for 500 iterations after which reasonable convergence was reached. Afterwards the mass flows of the different inlets have been noted and used for the real (compressible) simulation with the flow rates set at the inlets. There are several reasons why it would be preferable to set the flow rate as a negative value on the outlet. The major one is that this is the way the flow is initiated when the air induction systems are used in conjunction with an engine. It also gives a better comparison with the experimental results since the flow is created by sucking air through the system. It would also make the extra step of finding the division of the flow between the inlets unnecessary.

Solver settings and numerical schemes

As a start the corresponding settings as used in the Fluent computational procedure were tested. These did not provide enough numerical stability in OpenFOAM and a big part of the project has been finding settings that do. These are presented in section 4.1.

Air filter

The air filter is basically a wrinkled paper designed to filter the air while at the same time being cost-effective with long service intervals. To model the resistances in different coordinate directions the filter is modelled as a porous material. This is done using the Darcy-Forcheimer equation which means that a source term is added to the Navier-Stokes equations and the time derivative is attenuated (although this is irrelevant here because the air filter systems are simulated in a steady-state). Equation 3.2.1 together with source term 3.2.2 is the Darcy-Forchheimer equation where D_{ij} and F_{ij} specifies the viscous and inertial resistances in the

different coordinate directions.

$$\frac{\partial}{\partial t}(\gamma \rho v_i) + v_j \frac{\partial}{\partial x_j}(\rho v_i) = -\frac{\partial p}{\partial x_i} + \mu \frac{\partial \tau_{ij}}{\partial x_j} + S_i \tag{3.2.1}$$

The source term S_i is defined as:

$$S_i = -\left(\mu D_{ij} + \frac{1}{2}\rho |u_{jj}| F_{ij}\right) v_i \tag{3.2.2}$$

In *rhoPorousMRFSimpleFoam* the porous zone(s) is defined in the dictionary *porousZones*. It contains the name of the *cellZone* having the porous properties, the values of the diagonals of D_{ij} and F_{ij} as vectors and two vectors defining the coordinate system of the porous zone. The resistance values used are obtained from previous experiments at VCC. All of the simulations have been run with implicit porosity treatment since it increases numerical stability. For more information about the porous media implementation in OpenFOAM, please see [7].

3.2.2 Comparison with ANSYS Fluent

Since the equations upon which CFD rely are just models of reality, validation is very important. Validation means to compare the results of a simulation to experimental results and is the most important step in securing that the results obtained are of any use. In this thesis the OpenFOAM calculations are also compared against Fluent calculations as a first step. This is since Fluent has been used for a long time at the department and the results are known to usually be reasonably close to experimental data. The test cases were run with the following values for the boundary conditions:

Table 3.2.2: Boundary conditions for test cases

Boundary condition	Value
Velocity on inlet	Volvo reference mass flow rate
Pressure on outlet	Ambient pressure
Temperature on inlet	Ambient temperature
Mut roughness	0 roughness height (i.e. no roughness)

The two test cases used will be evaluated using a number of different measures that are of interest to VCC. These are presented below.

Streamlines coloured by velocity. Streamlines are a good way to get an idea of how the flow actually looks. If there are any unphysical effects or strange flow behaviours they can be seen here. The streamlines also show if the flow is, for example, swirling.

Pressure drop at all sections. The most important parameter to measure as an increased pressure drop will translate into a less efficient engine. The pressure drop is measured in relation to the total pressure at the inlet. The average on each section is taken with respect to mass weighting because the flow is compressible, and hence the density can differ on different

cell faces. The measurement sections are shown in Figure 3.2.3 and 3.2.4 for the two test cases respectively.

Uniformity index. The uniformity index is an industry standard measure of how uniform the velocity distribution of the flow is at a particular section. It is defined by Equation 3.2.3:

$$\gamma = 1 - \int_A \frac{\sqrt{(\overline{u} - u)^2}}{2A\overline{u}} dA \tag{3.2.3}$$

The sections where flow uniformity is of special interest for the air filter systems are:

- The surface of the air filter. A uniform flow here is important for filter utilisation.
- In front of the air mass meter. Important with a high flow uniformity to ensure correct signal from the sensor.
- The outlet, for even distribution to the engine.

Visual representation of flow distribution at sections. A visual representation of the flow distribution is also shown to get an idea of where the possibly uneven velocities are located on the surface.

To extract total pressure and uniformity index on the internal sections in OpenFOAM the swak4foam extension has been used. As described in section 3.1.6 swak4foam is among other things an extension to the functionObjects in OpenFOAM. The functionObjects execute a piece of code on each iteration/time-step and as such the total pressure and uniformity index can be read at every iteration. To produce images with data from the internal surfaces (*faceZones* in OpenFOAM) the only way known to the author is to use Paraview with the paraFoam wrapper, which is supplied with OpenFOAM. Data not belonging to the internal surfaces can be converted to a range of formats but Paraview has been used throughout for the OpenFOAM results.

3.2.3 Test case one: I5D P1

This is the air induction system found in Volvos small car platform (P1) in conjunction with the 5 cylinder diesel engine. It has two inlets on the dirty side (before the flow passes the filter) and one outlet. Figure 3.2.3 shows the model with the measurement sections highlighted in black.

The mesh consists of approximately 6.8 million cells which makes it one of the larger used at VCC for this type of simulation.

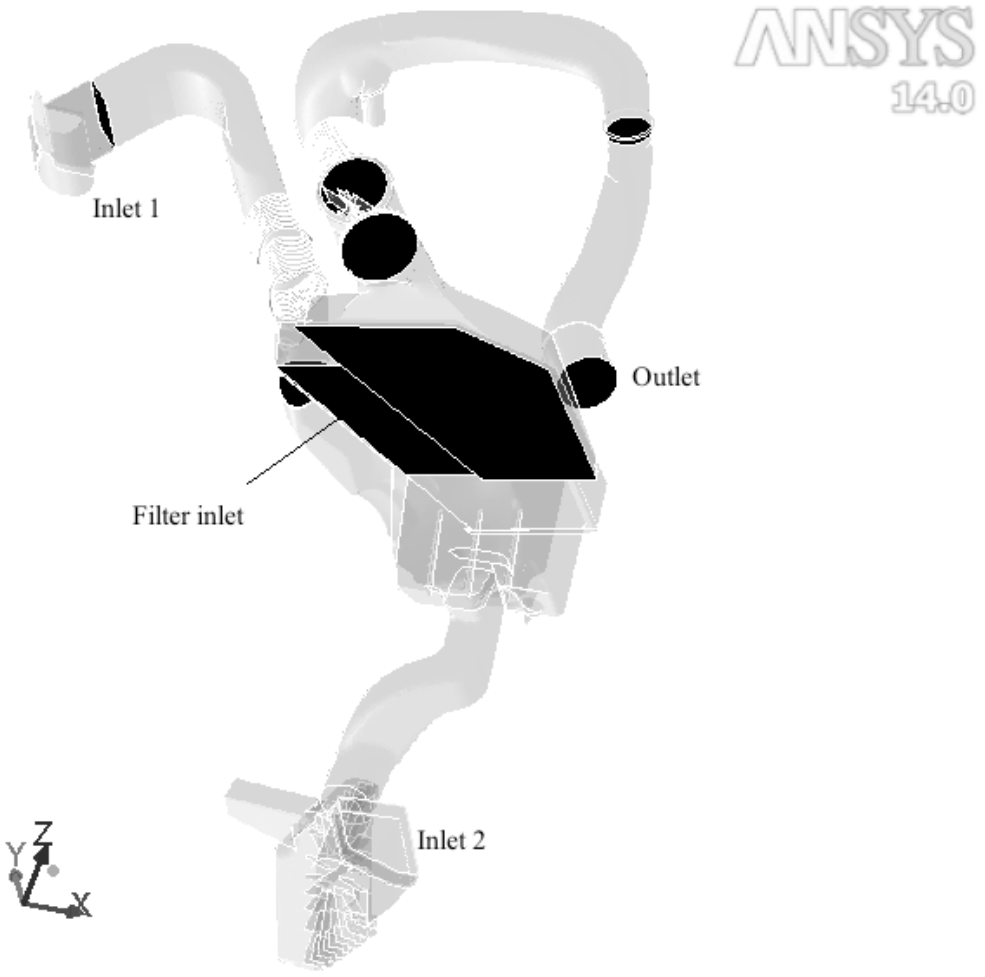


Figure 3.2.3: *I5D P1 model with highlighted measurement sections*

3.2.4 Test case two: I5D EuCD

The model used here is that of the air induction system for Volvos 5 cylinder diesel engine in the EuCD Platform (models V70, S80). Figure 3.2.4 shows the model with the measurement sections highlighted in black.

The number of cells in this model totals a little over 6 million. The main reason for it being smaller is that the geometry is physically smaller. It has two inlets in the same way as case one, but one of them is only a hole in the filter box and the other one is connected with a shorter duct.

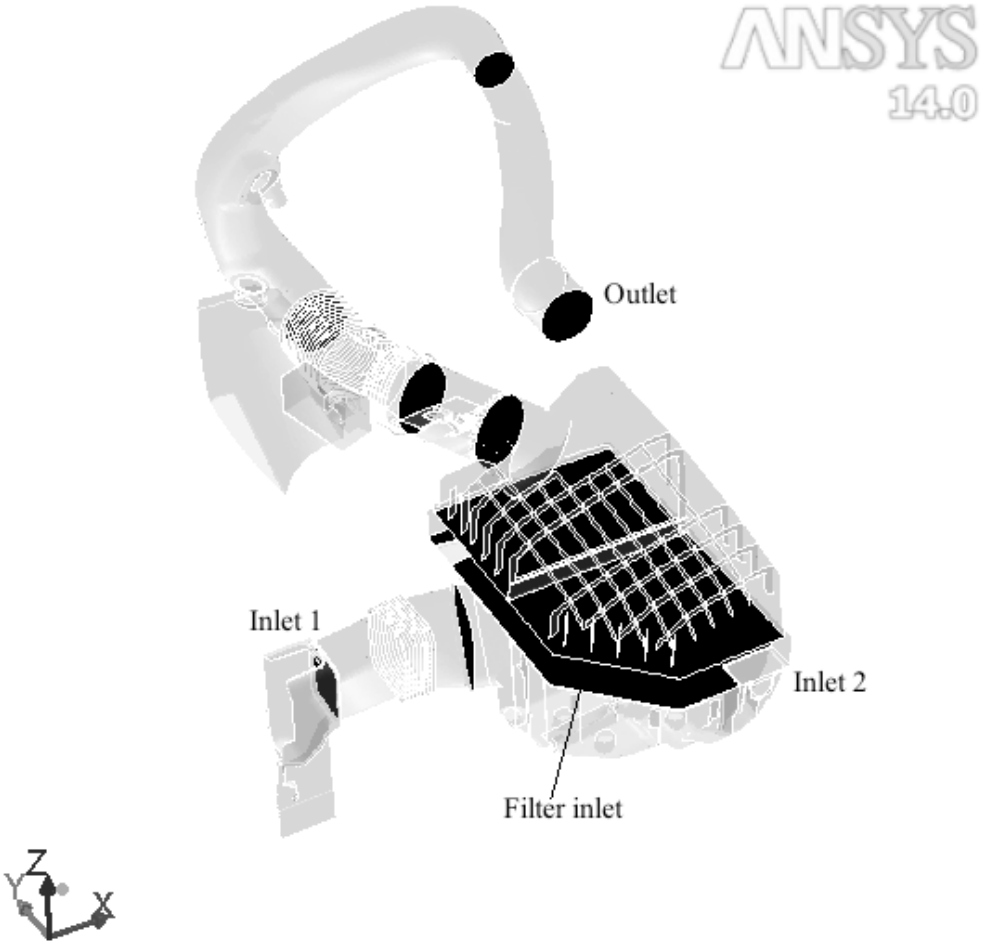


Figure 3.2.4: *I5D EuCD model with highlighted measurement sections*

3.2.5 Validation with experiments

It is not enough to compare to another CFD software, because either could be the more accurate one. Therefore the pressure drops obtained from the two CFD softwares are compared to measurements from the VCC flow rig. The rig measures the total pressure drop from inlet to outlet. Hence it is only one value to compare against, but it still serves as an indication on which software is closest to the real values. Figure 3.2.5 shows the setup of the flow rig with the system pertaining to the P1 platform in place.

3.3 Topology optimisation

As mentioned in section 1.1 the single biggest constraint in the design of the air induction systems is packaging. The topologies of the air induction systems need to be optimised mainly for pressure loss and for uniformity at certain sections. There are a number of optimisation methods such as using a DOE to test how variations in certain parameters affect the result (basically trial and error), gradient based methods or by just using the experience of the designer.

Figure 3.2.5: *I5D P1 air induction system mounted in the flow rig*

A previous thesis work [8] has shown how to use the optimisation software modeFrontier in the optimisation process for air induction systems. ModeFrontier uses a DOE to test how variations in parameters affect the variables optimised against. Even though a different CFD-solver was used in [8] (STAR-CCM+), the workflow would be the same with OpenFOAM. And since OpenFOAM is command-line based it is very easy to integrate with modeFrontier. Instead of using a DOE-approach to optimisation this thesis has focused on using the adjoint method.

3.3.1 Adjoint method

The adjoint method is a gradient based method. It has been widely used for some time in the aerospace industry, but has fairly recently been introduced within the automotive industry. It is a method which evaluates the derivative of a function $I(w, F)$ with respect to F even if $I(w, F)$ only depends on F indirectly via an intermediate variable w [9]. The derivation of the basic equations below follow [9] and [10].

Starting with a scalar function I

$$I = I(w, F) \tag{3.3.1}$$

where w are flow field variables and F are grid variables.

Differentiate the cost function once to obtain:

$$\delta I = \frac{\partial I}{\partial w}\delta w + \frac{\partial I}{\partial F}\delta F \tag{3.3.2}$$

The flow field is:

$$R(w, F) = 0 \tag{3.3.3}$$

And the first perturbation:

$$\delta R = \frac{\partial R}{\partial w}\delta w + \frac{\partial R}{\partial F}\delta F \tag{3.3.4}$$

Since $\delta R = 0$ [9] it can be multiplied with an arbitrary term Ψ. Subtracting this from 3.3.2 gives:

$$\delta I = \frac{\partial I}{\partial w}\delta w + \frac{\partial I}{\partial F}\delta F - \Psi\left(\frac{\partial R}{\partial w}\delta w + \frac{\partial R}{\partial F}\delta F\right) \tag{3.3.5}$$

If Ψ is chosen so that:

$$\frac{\partial R}{\partial w}\Psi = \frac{\partial I}{\partial w} \tag{3.3.6}$$

The derivative of $I(w, F)$ may be calculated without taking the derivative with respect to w as:

$$\delta I = \left(\frac{\partial I}{\partial F} - \Psi\frac{\partial R}{\partial F}\right)\delta F \tag{3.3.7}$$

Equation 3.3.6 is called the adjoint equation and this together with 3.3.7 is called the discrete adjoint method. The adjoint method calculates the derivative of the cost function with respect to grid variables, but doesn't model the variables. In other words, it models the sensitivity of the cost function to some chosen variables. The cost function can be chosen so that it reflects the quantity that is optimised against. The computational cost doesn't depend on the number of objectives solved for, since it just means incorporating them into the cost function.

3.3.2 Adjoint method and OpenFOAM

There is already one solver distributed with OpenFOAM utilising the adjoint method called *adjointShapeOptimizationFoam*. It is a steady-state solver for incompressible turbulent flows. It optimises a geometry for low pressure loss by blocking cells where the sensitivity to pressure loss is high. The blocking is achieved by adding an individual porosity to each cell via Darcy's law (see section 3.2.1). The optimised topology can then be extracted as the boundary between

cells with porous blockage and no blockage. The existing solver optimises the whole domain it is given, and as such has no means of treating some parts as "frozen".

To test the solver a fictitious design space was used. As a baseline design a constant radius pipe was inserted. At the thinnest section of the design space the pipe was removed and a volume utilising the maximum available packaging space was inserted. In this box the adjoint solver can create blockages to optimise the geometry against pressure loss. The resulting geometry is showed in figure 3.3.1. A mass flow was set at the inlet and a static pressure at the outlet. The adjoint velocity and pressure used the required adjoint boundary conditions *adjointOutletVelocity* and *adjointOutletPressure* on the outlet. Also, the adjoint velocity was set to a negative value at the inlet. The approach is to compare the results before and after the adjoint solver has blocked some of the cells in the design space and see if the pressure losses have been lowered.

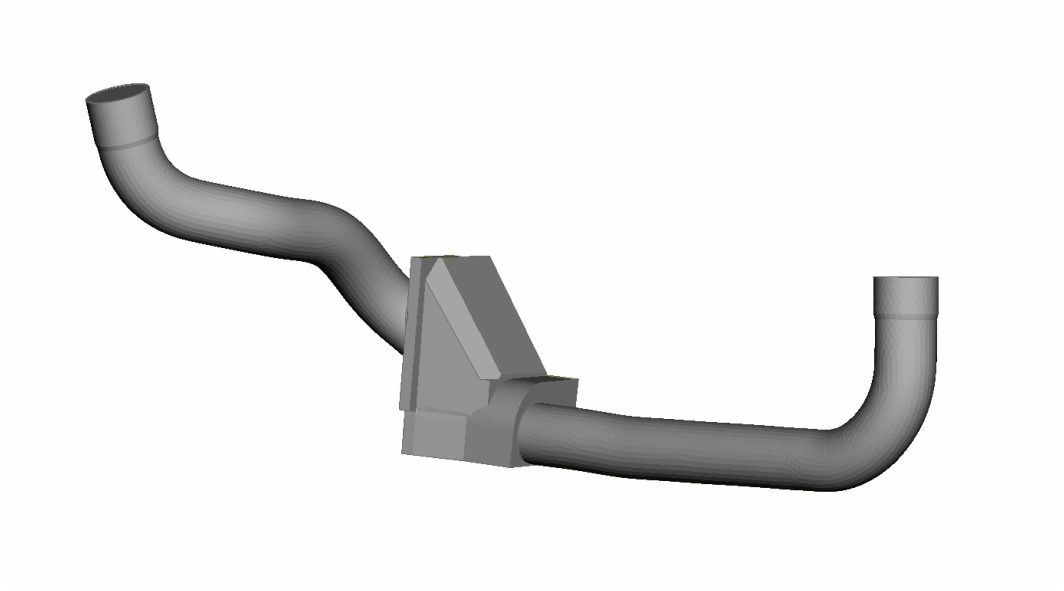


Figure 3.3.1: *Geometry used when testing the OpenFOAM adjoint solver*

4 Results

This chapter starts out with presenting the OpenFOAM settings found to work best for the air induction system simulations. The next part contains results from the comparison between Fluent and OpenFOAM and also to experimental values. A small study of the parallel scaling in OpenFOAM is also presented and last there is a small evaluation of the OpenFOAM adjoint solver.

4.1 OpenFOAM settings

The first approach was to try and mimic the computational procedure for Fluent, and to achieve this a Python script was developed which performs the switch from incompressible to compressible simulation and also changed discretization schemes at certain intervals. In OpenFOAM it is a little bit more involved to automatically change from incompressible to compressible settings. The solver has to be changed, the dissipation ϵ and turbulent kinetic energy k needs different wall functions and the pressure and flux phi have different dimensions. This approach was abandoned mainly because of stability problems and also because it was unnecessarily complicated for day-to-day usage. Although not used for these simulations the script is attached in Appendix C as an alternative way to initialise the simulations. Instead of the incompressible to compressible switch a different approach for achieving a stable solution was used. The air induction system simulations are very sensitive to numerical settings. None of the two tested models could be run with all discretization schemes set to second order. Despite a great number of combinations of settings for under relaxation factors, linear solvers and tolerances etc tested or by using methods such as starting with first order schemes, starting with incompressible simulations or ramping up the velocity gradually made for stable solutions. Since none of the tested methods made the simulation converge with all second order discretization schemes some had to be only first accurate, which will implicate the results. Even though not all of the fields could use the increased accuracy of second order schemes, the momentum field could. Since this is the most important field, the effects on accuracy should not be too severe. All of the above mentioned methods were able to start the simulations, and the easiest to use is ramping up the velocity and as a consequence it was used for the comparison cases. The flow rate was set to increase every 200th iteration during the first 1000 iterations where the full flow rate was set.

The solver settings specified in the *fvSolution* dictionary used for the comparison runs were:

```
solvers
{
    p
    {
        solver                  GAMG;
        tolerance               1e-8;
        relTol                  0.001;
        smoother                GaussSeidel;
        cacheAgglomeration off;
        nCellsInCoarsestLevel 20;
        agglomerator            faceAreaPair;
        mergeLevels             1;
    }
    U   //not needed when using implicit porosity treatment
    {
        solver          smoothSolver;
        smoother                GaussSeidel;
        nSweeps         2;
        tolerance               1e-7;
        relTol                  0.01;
    }
    "(k|epsilon)"
    {
        solver          smoothSolver;
        smoother                GaussSeidel;
        nSweeps          2;
        tolerance               1e-10;
        relTol                  0;
    }
    h
    {
        solver          smoothSolver;
        smoother                GaussSeidel;
        tolerance               1e-07;
        relTol          0.0001;
    }
}
SIMPLE
{
        nUCorrectors    2;
        nNonOrthogonalCorrectors 2;
        rhoMin   rhoMin [ 1 -3 0 0 0 ] 0.5;
        rhoMax   rhoMax [ 1 -3 0 0 0 ] 1.5;
}
```

With under relaxation factors as presented in Table 4.1.1

Table 4.1.1: Under relaxation factors

Field	URF
p	0.1
rho	0.9
U	0.8
k	0.7
epsilon	0.7
h	0.7

Worth noting is that lowering the under relaxation factor for the momentum U results in not just a slower solution but decreases stability as well. The tolerances of the solvers have been set low to lower the residuals, even though it increases solution time.

Discretization schemes used were the default ones except for divergence, where the second order accurate scheme *linearUpwind(V) cellMDLimited Gauss linear 1* was used for div(phi,U) and div(U,p) while the first order accurate *upwind* scheme was used for the rest.

4.2 Fluent - OpenFOAM comparison

4.2.1 Case one: I5D P1

The simulation was run for 3000 iterations, after which the highest residual (pressure) was 10^{-3}. The mesh contained a few non-orthogonal cells and because of that *nNonOrthogonalCorrectors* was set to 2. This improved the pressure residual one order of magnitude. The PCG (*Preconditioned Conjugate Gradient*) solver was also tested with this case but was found to be a lot slower and also caused the solution to diverge after about 800 iterations, i.e. even before the full flow rate had been reached. The resulting flow is pictured in figure 4.2.1 and for comparison the model calculated with Fluent is shown in figure 4.2.2. The streamlines in both plots are tracked in reverse from the outlet since there is more than one inlet. The flows look similar with approximately the same velocities and swirls in the dirty side duct (before the air filter) and at the outlet. One thing to notice is that fewer of the streamlines seem to make it through the air filter box in the OpenFOAM simulation (as tracked from the outlet).

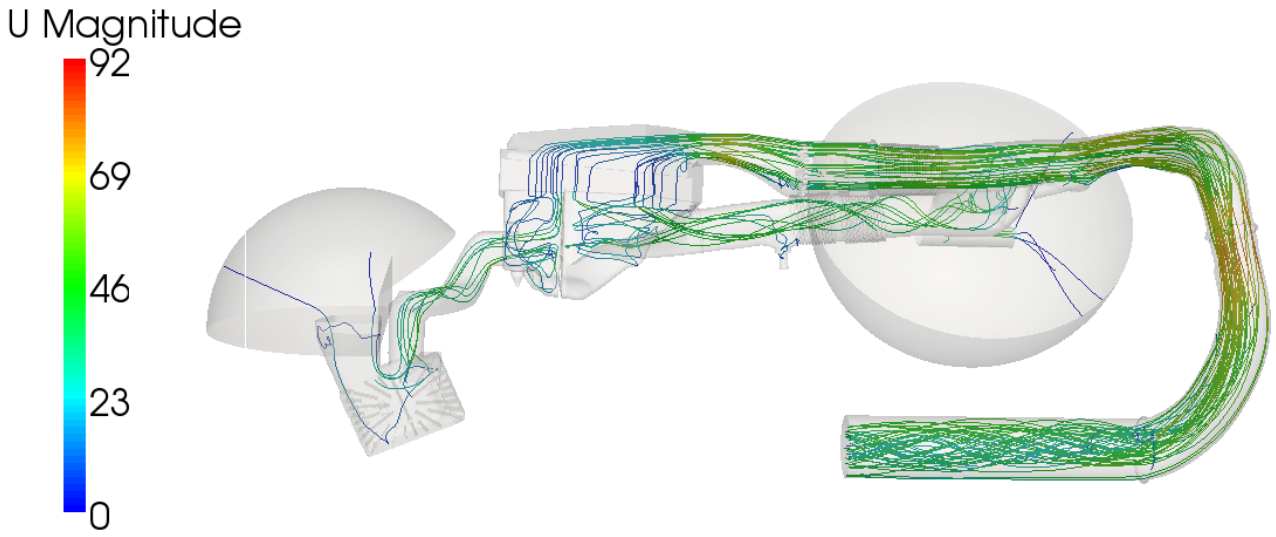


Figure 4.2.1: *Streamlines in the P1 system as calculated with OpenFOAM*

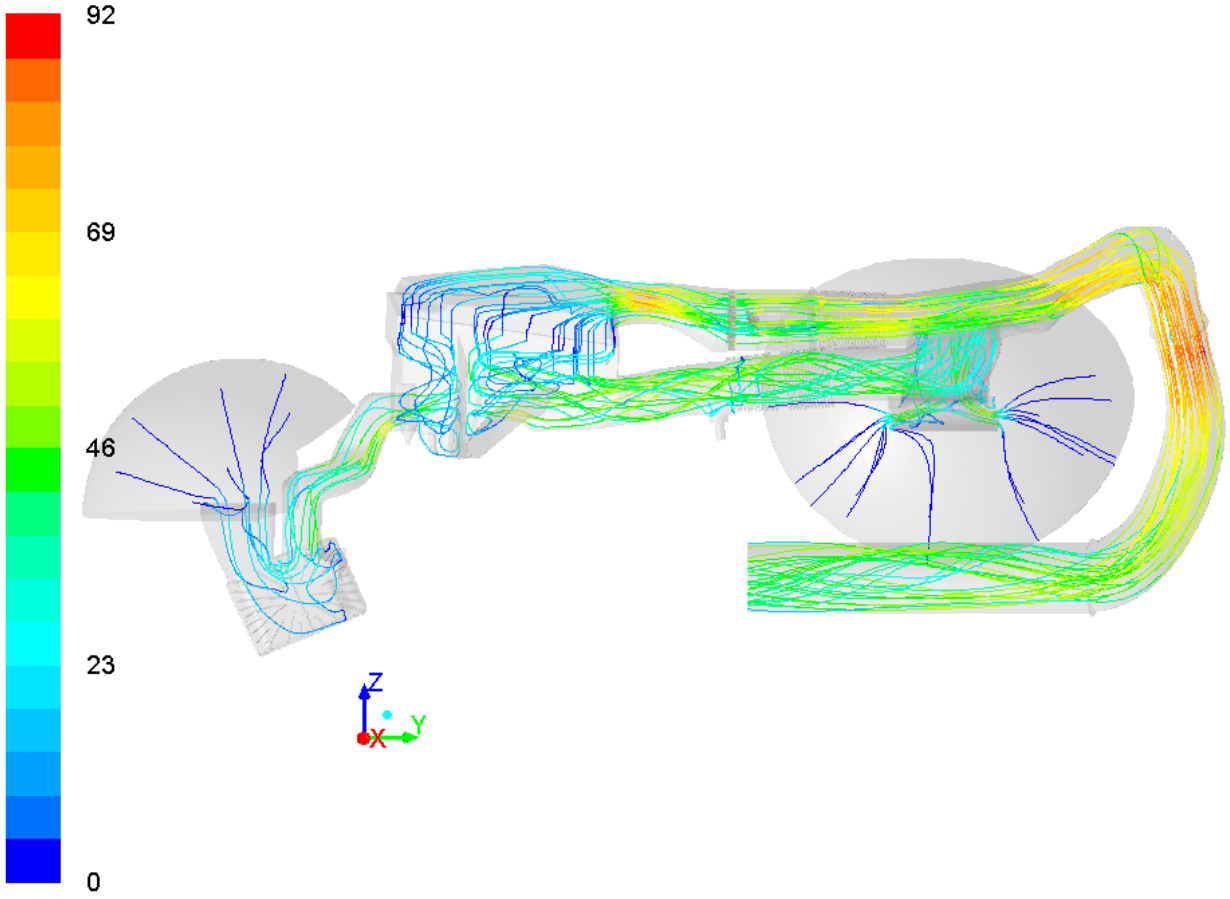


Figure 4.2.2: *Streamlines in the P1 system as calculated with Fluent*

To study the pressure drop in more detail, the absolute values for total pressure drop from the inlet are presented in Table 4.2.1 together with corresponding values from Fluent. The

pressure drop and uniformity index from OpenFOAM are calculated with the swak4foam script in Appendix B, while the Fluent values are calculated with built-in Fluent methods.

Table 4.2.1: Mass weighted average of total pressure drop (Pa)

	Fluent	OpenFOAM	Difference
sec2_entrance	-893	-1248	39.8%
sec3_dsd_out	-1623	-1662	2.4%
sec4_filter_inlet	-2752	-2878	4.6%
sec5_filter_out	-3115	-3165	1.6%
sec6_amm_in	-3481	-3418	-1.8%
sec7_csd1_in	-4366	-4121	-5.6%
sec8_csd2_in	-5449	-5185	-4.8%
sec9_outlet	-5903	-5515	-6.6%

The first thing to note is that the OpenFOAM solver predicts the intake losses as a lot higher. This could be down to the difference in the way the boundary conditions are applied. OpenFOAM has been unable to achieve a stable solution with a negative mass flow on the outlet, and instead the mass flow has been set at the inlets. When going downstream the pressure losses are much more similar in magnitude between the two softwares, and just after the filter OpenFOAM starts to predict a lower pressure loss than Fluent. That is, OpenFOAM predicts a higher intake loss but a lower loss for the flow through the ducts. Figure 4.2.3 shows the filter inlet velocity distribution for Fluent and OpenFOAM. The velocity distributions are relatively similar but the solution from OpenFOAM is more chaotic in its nature. As will be shown later (section 4.2.3) OpenFOAM has problems with the velocities at the filter inlet. Even so, the correlation here is good compared to case two (figure 4.2.7). At the outlet (figure 4.2.4) the velocity distributions are similar from both codes, as they should be.

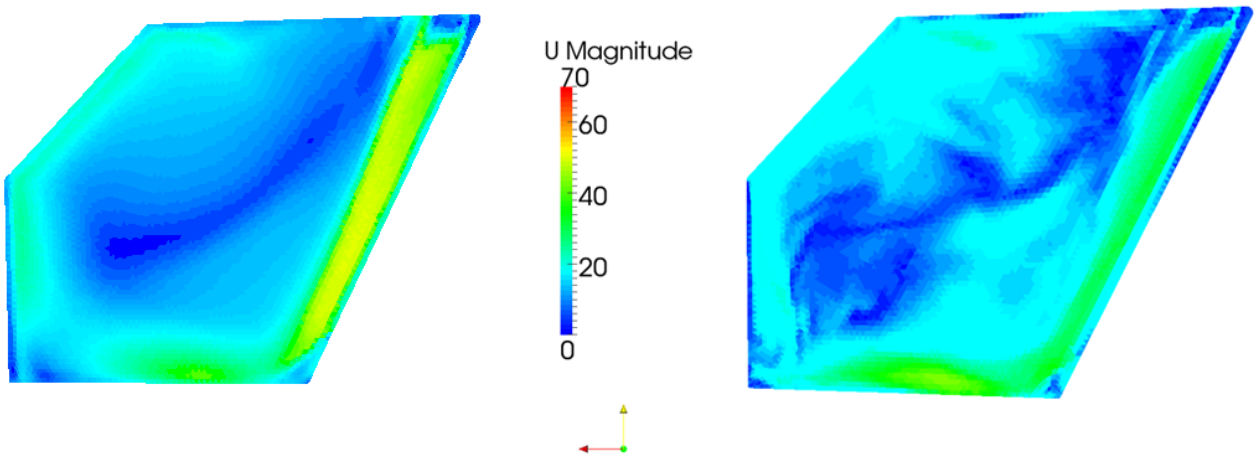


Figure 4.2.3: *Velocity distributions at the filter inlet in the P1 system with Fluent on the left and OpenFOAM on the right*

The uniformity indexes (table 4.2.2) are similar at the outlet and before the air mass meter, but the figure that stands out is at the filter inlet. This is not only a bad correlation to Fluent, which is not necessarily "correct", but the log files indicate that the uniformity index at the

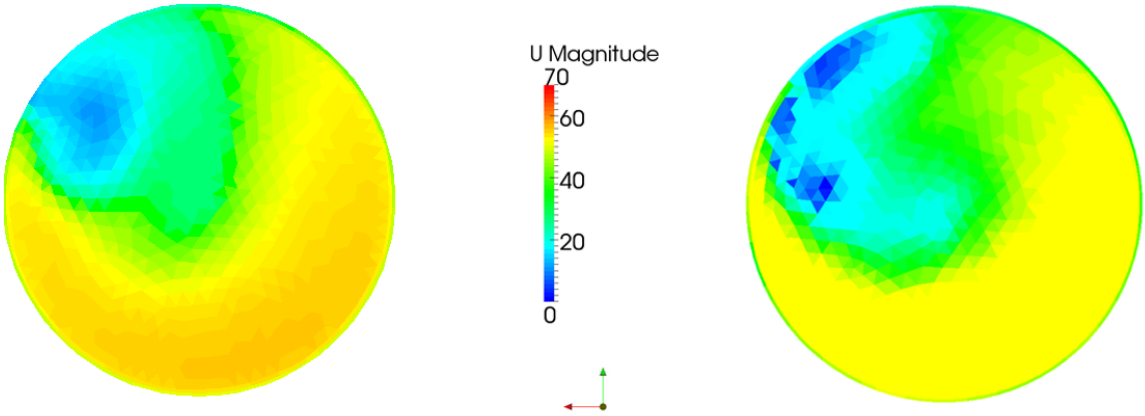


Figure 4.2.4: *Velocity distributions at the outlet of the P1 system with Fluent on the left and OpenFOAM on the right*

filter inlet varies between iterations. The other two uniformity indexes are constant, indicating that there is a problem with the velocity field at the filter inlet. This phenomenon is studied in more detail in section 4.2.3.

Table 4.2.2: Area weighted uniformity index

	Fluent	OpenFOAM	Difference
sec4_filter_inlet	0.763	0.854	11.9%
sec6_amm_in	0.924	0.948	2.6%
sec9_outlet	0.869	0.854	-1.7%

4.2.2 Case two: I5D EuCD

The geometry of case two is simpler than that of case one with no duct connecting inlet two to the air filter box and a shorter duct from inlet 1. With the same level of mesh refinement this led to a shorter solution time of just under 2 hours for the same amount of iterations as in case one. As in case one the highest residual (pressure) was in the region of 10^{-3}. Also in this case both softwares capture the outlet swirl in the same way (figures 4.2.5 and 4.2.6). Interesting to note is that in this case as well a smaller part of the streamlines (tracked in reverse direction) seem to make it past the air filter. The reason for this is unknown, but could be related to the phenomenon described in section 4.2.3.

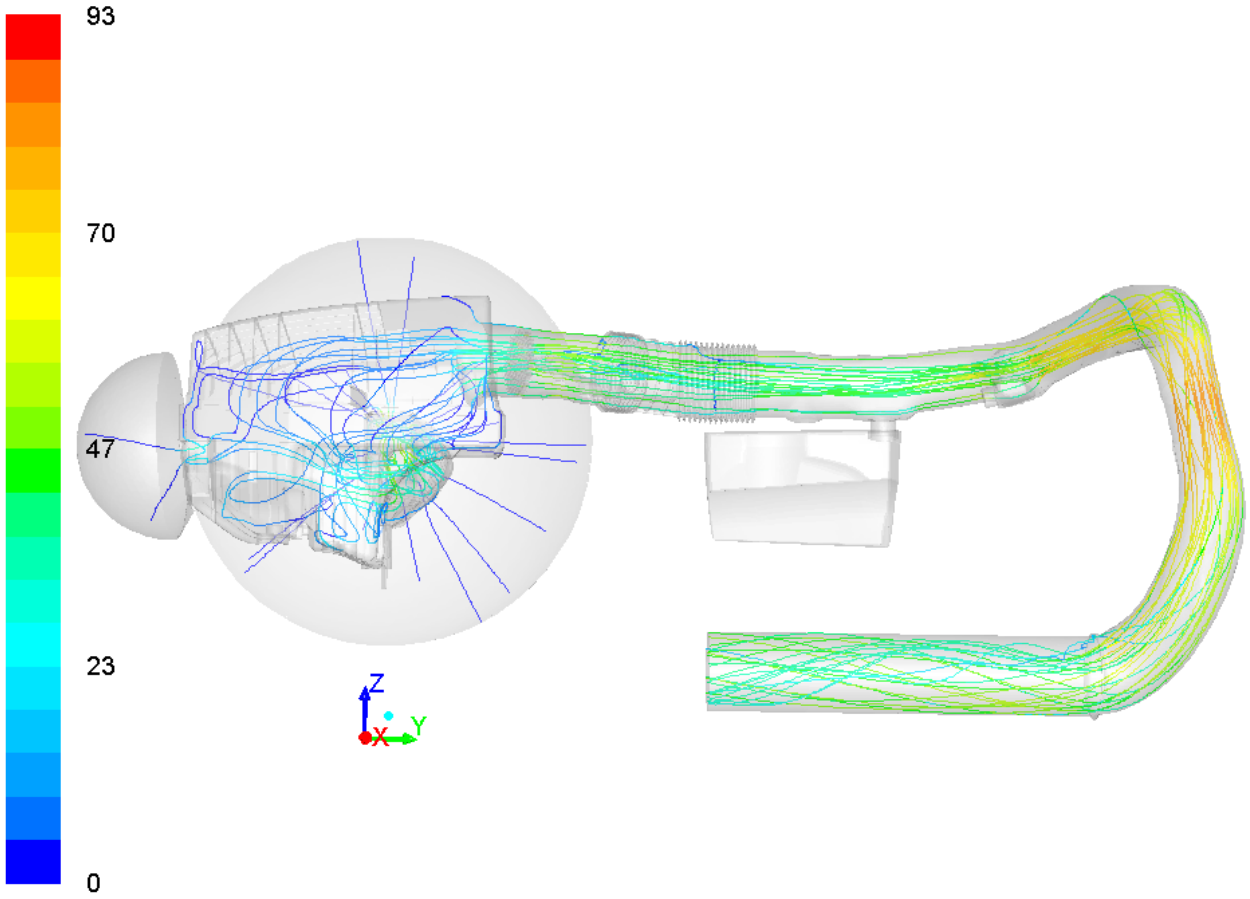


Figure 4.2.5: *Streamlines in the EuCD system as calculated with Fluent*

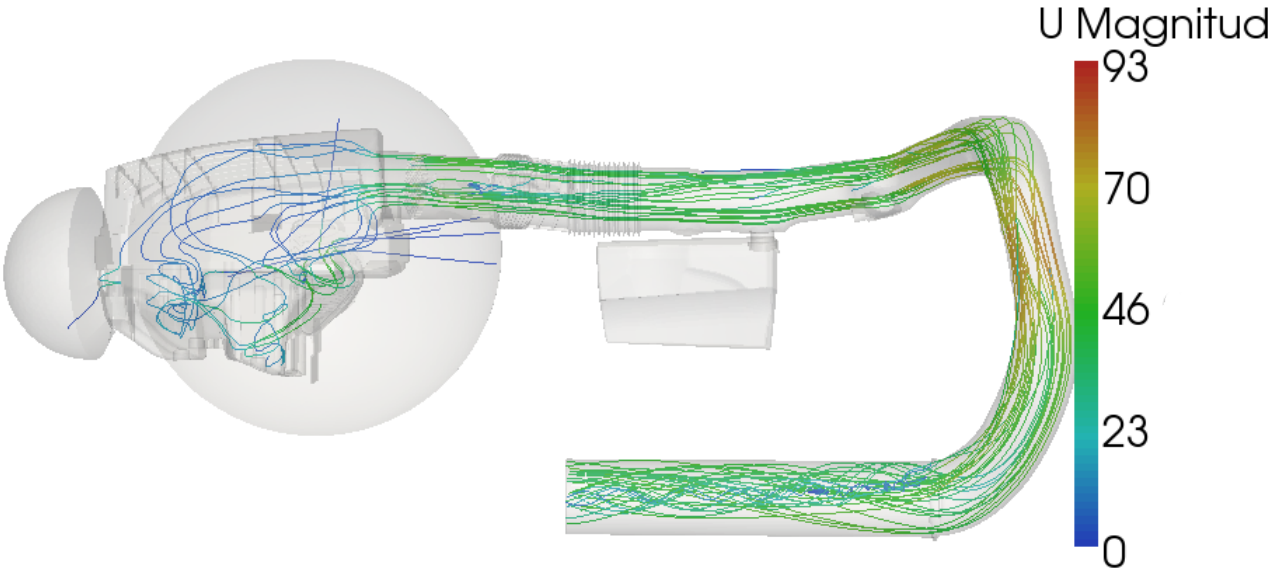


Figure 4.2.6: *Streamlines in the EuCD system as calculated with OpenFOAM*

Also in this case the same trend with a prediction of a much higher pressure loss in the intake but then a lower loss in the rest of the system can be observed (Table 4.2.3). Worth noting is that in this model OpenFOAM only predicts a pressure loss of 44 Pa for the air filter, something which is very low compared to the Fluent simulation. In case one the pressure loss over the air filter is almost 300 Pa despite the same viscous and inertial resistances and the same mass flow in both systems. One possible reason for the low pressure drop over the air filter is that it is the total pressure that is measured. The total pressure has been calculated as $p_0 = \frac{1}{2}\rho U^2 + p$ and since it includes the velocity it can be affected by the oscillating velocities seen in the interface to a porous zone.

Table 4.2.3: Mass weighted average of total pressure drop (Pa)

	Fluent	OpenFOAM	Difference
sec2_dsd_in	-234	-289	23.5%
sec3_dirtysideduct_out	-555	-708	27.6%
sec4_filter_inlet	-1101	-1245	13.1%
sec5_filteroutlet	-1293	-1289	-0.3%
sec6a_amm_in	-1401	-1522	8.6%
sec6b_amm_out	-2120	-2178	2.7%
sec7_csd2_in	-2355	-2394	1.6%
sec8_csd3_in	-2968	-2744	-7.5%
sec9_outlet	-3430	-3130	-8.8%

Next, velocity distributions at the filter inlet and system outlet are compared. It is evident that the velocity distribution at the filter inlet differs quite a lot between Fluent and OpenFOAM for the EuCD model (Figure 4.2.7). The outlet shows a good correlation between the two softwares (Figure 4.2.8).

As in case one the uniformity index at the filter inlet differs significantly between the two softwares while at the outlet and in front of the AMM they are similar. Also in this case the uniformity index at the filter inlet oscillates as if the solution was transient. The rest of the flow is steady indicating unphysical velocity patterns at the interface of the porous zone.

Table 4.2.4: Area weighted uniformity index

	Fluent	OpenFOAM	Difference
sec4_filterinlet	0.787	0.940	19.4%
sec6_amm_in	0.959	0.942	-1.7%
sec9_outlet	0.907	0.890	-1.9%

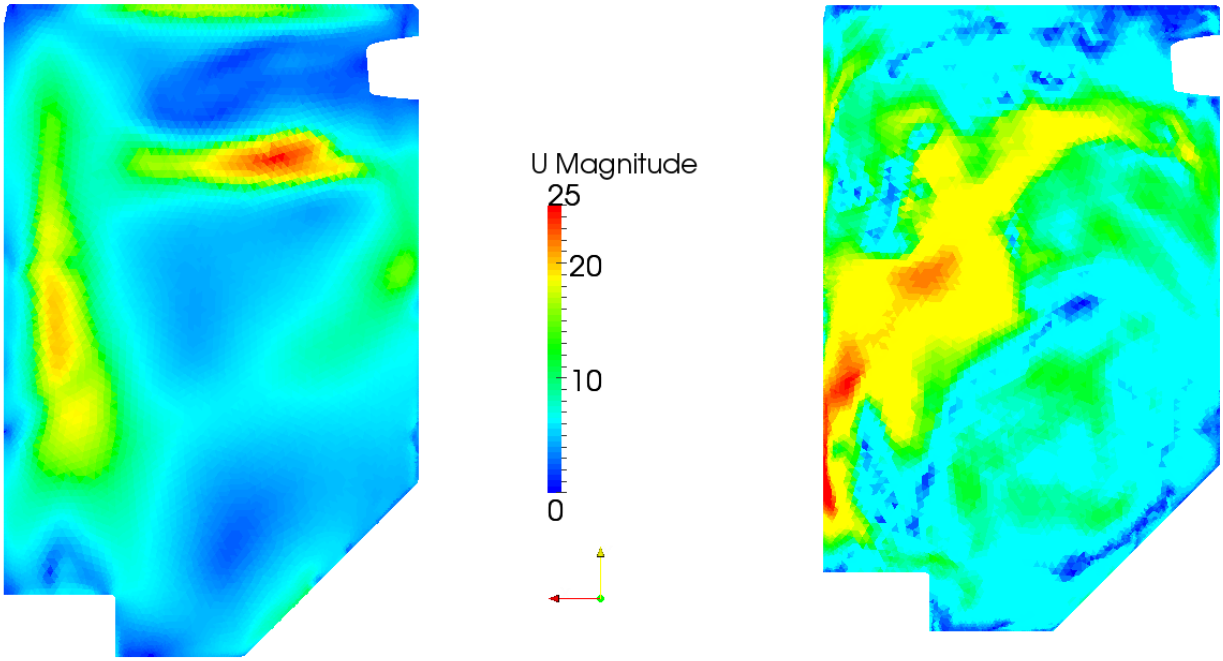


Figure 4.2.7: *Velocity distributions at the filter inlet in the EuCD system with Fluent on the left and OpenFOAM on the right*

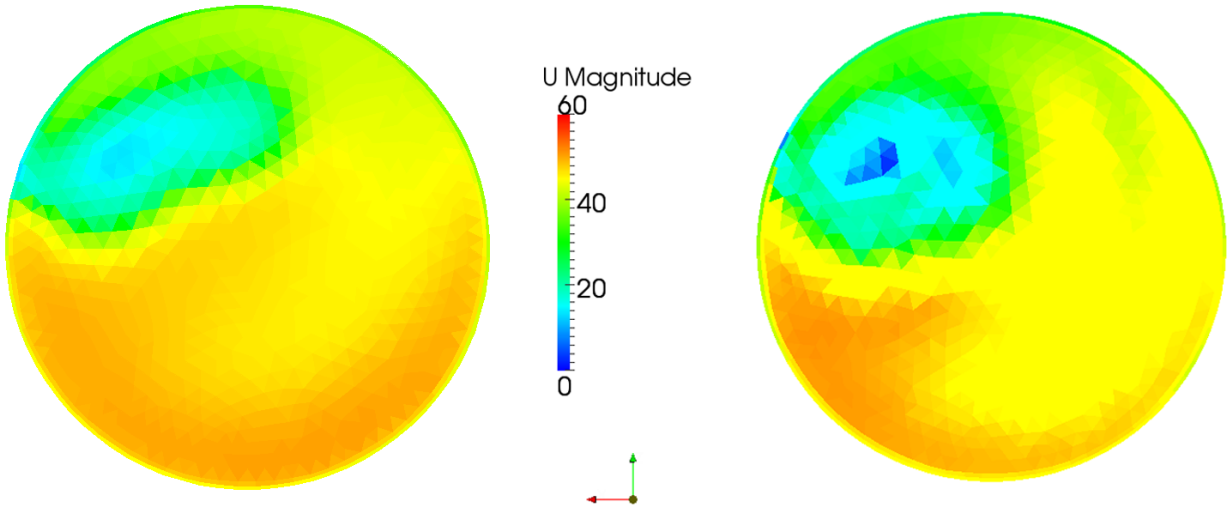


Figure 4.2.8: *Velocity distributions at the outlet of the EuCD system with Fluent on the left and OpenFOAM on the right*

4.2.3 Oscillating velocity at porous interface

The uniformity indexes at the filter inlet as calculated with OpenFOAM are significantly different in both the test cases from the ones calculated with Fluent. This prompted further investigation and it was found that it also varied between subsequent iterations. The rest of the flow is steady, and it is just around the interface between non-porous/porous media that this behaviour with an oscillating velocity can be observed. Figure 4.2.9 shows how the velocities in the three coordinate directions vary along a straight line thru the porous zone (air filter). The porous zone itself starts at a curve length of ≈ 0.037 and ends at ≈ 0.074. The plot is from test case two, but the same behaviour has been observed in the first case as well. The velocity in x-direction goes towards zero as it enters the porous zone, which is as expected since this is the direction with the highest resistance. In the Y-direction, which is the direction along the filter paper ribs, the oscillation can be observed. At the start of the filter the velocity fluctuates between positive and negative values. Also the Z-velocity shows oscillations, but stays at positive values. These oscillations are not visible in the data from the Fluent simulation.

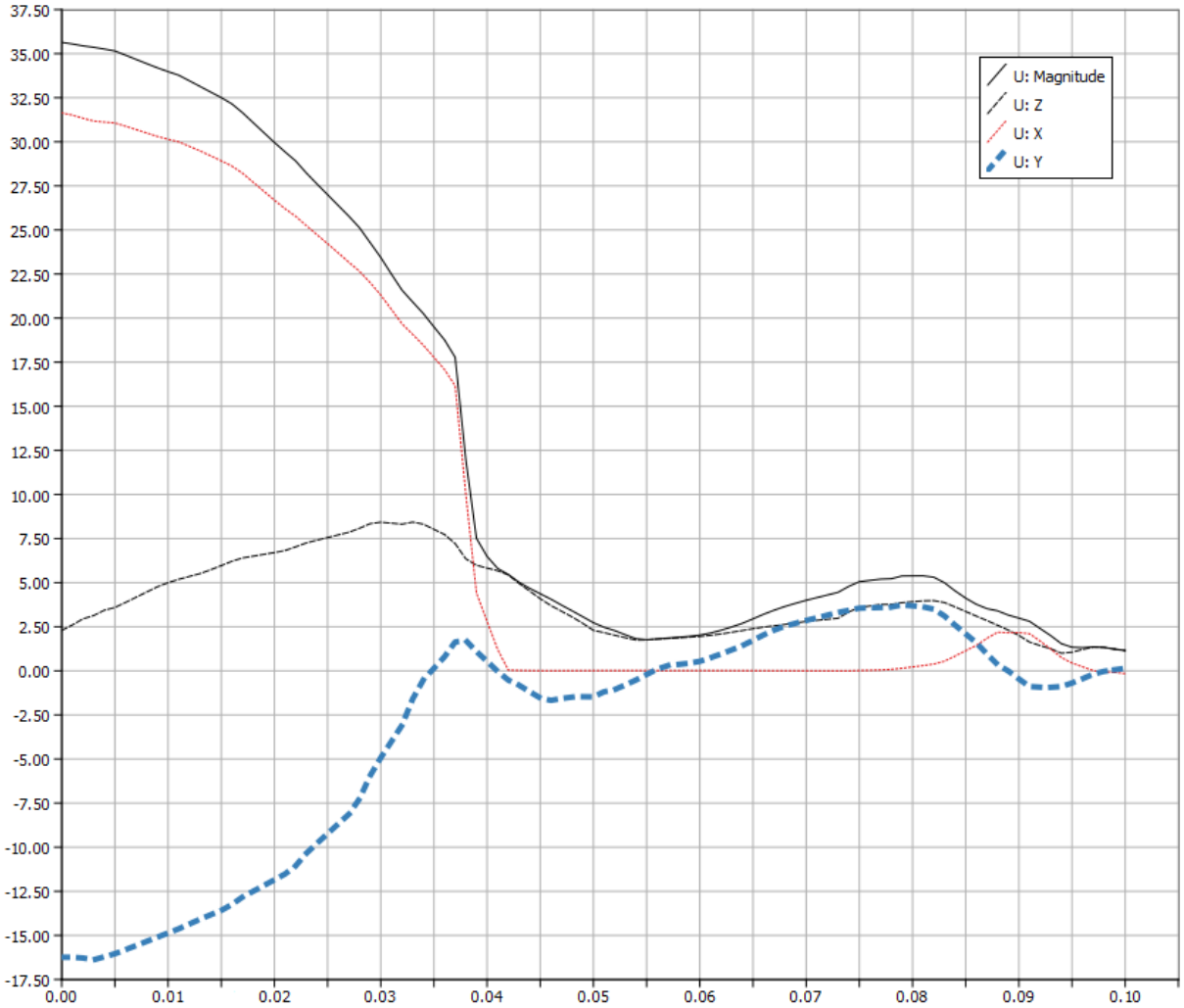


Figure 4.2.9: *Velocity profile through the filter as computed with OpenFOAM*

A search in the OpenFOAM bug reporting system shows that this problem is a known one

[11]. In the bug report there is also a suggested fix. It consists of replacing the line of code *U -= rUA*fvc::grad(p)* for *U = fvc::reconstruct(phi)/rho* in the pEqn.H file used in the *rhoPorousMRFSimpleFoam* source file. This change was made and the solver recompiled. Although this is somewhat outside of the scope set up in the limitations (section 1.3) it was deemed important for the conclusions to evaluate the fix. Both of the test cases were run again with the new solver but otherwise identical settings. The results were similar to the original solver and the velocity patterns remained unphysical and changing on every iteration at the start of the porous zone.

4.3 Validation with experiments

Total pressure drops over the systems as measured from experiments are presented in Table 4.3.1. For these cases the OpenFOAM simulations are further from the experimental results. In the P1 case OpenFOAM is off by 9% and in the EuCD case by a substantial 25%, compared to 2 and 18% for Fluent. The measurements in the flow rig are taken by a pressure sensor mounted between the outlet and the attachment to flow rig, visible in figure 3.2.5. One thing to have in mind during this comparison is that OpenFOAM could not be made to run with the flow set at the outlet, and even though the mass flows through the systems are the same, the boundary conditions are not identical.

Table 4.3.1: Total pressure drop computed from CFD simulations and measurements (Pa)

Case	OpenFOAM	Fluent	Experiment
P1	5515	5903	6052
EuCD	3130	3430	4177

4.4 Parallel scaling

Since CFD is very demanding with respect to computing power, it is important to use the optimal number of processors/cores when doing an analysis utilising parallel computing. This is important both in respect to shortening lead times in development projects and to use the available computing resources as efficiently as possible. OpenFOAM uses domain decomposition for processing in parallel. This means that the geometry and associated fields are divided into parts for each processor/core going to be used for the computation. To determine the optimal number of cores to use when running cases in parallel, a test has been conducted by solving the same case using a varying amount of cores. The model used is similar to the test cases and consists of 6 million cells. This is a typical size for air induction system models. The type of simulation is the same as in the test cases, i.e. steady state, compressible with realizable k-epsilon turbulence model and porous media for the filter. To obtain a simulation time, the case was run for 2000 iterations and the clocktime read from the log file. The case was run with 16,32,48,64,96,128,144,160,172,192, 256 and 384 cores. The number of cores used is only relevant for a case with around the same number of cells as the one used in this test. Instead, the average number of cells per core has been used as a measure to help decide the

optimum number of cores for any given case. Presented in figure 4.4.1, the conclusion is that the optimum lies around 50000 cells per core used. With more cells than this per core the interconnections aren't saturated and so more cores equals shorter solution time. And with fewer cells per core, the interconnections between the cores/nodes become the bottleneck and this might actually lead to a slower calculation.

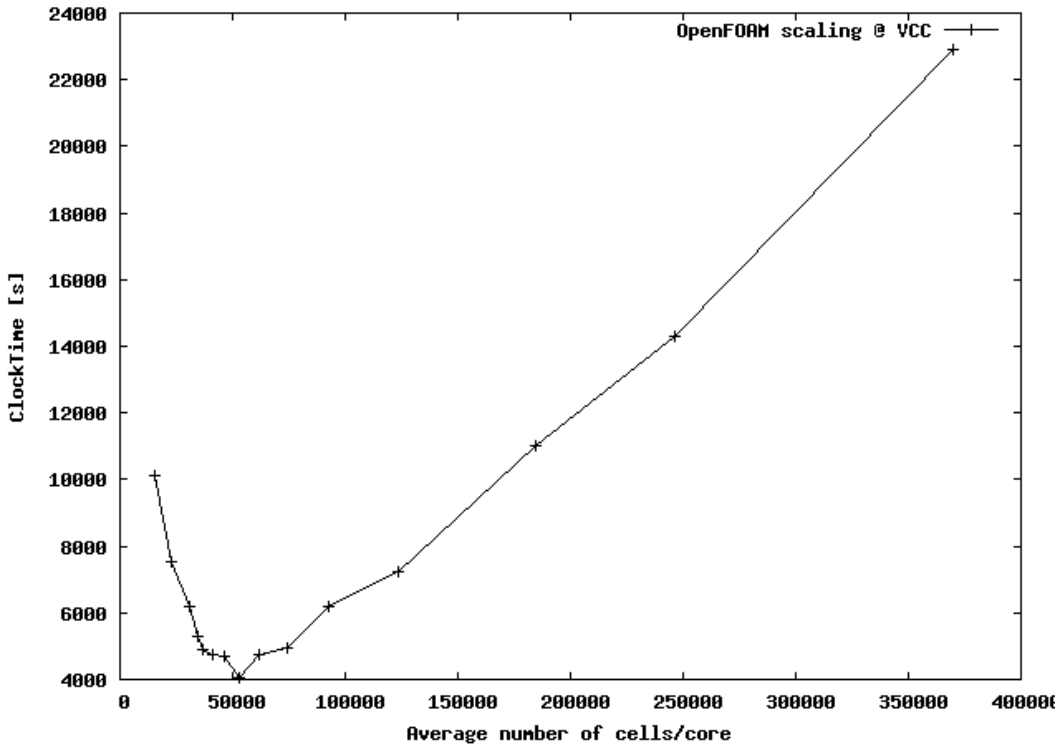


Figure 4.4.1: *Number of cells/core for optimum computing performance*

4.5 Topology optimisation

To achieve convergence with the adjoint solver proved to be difficult, but after raising the value *alphaMax* to 10^6, which corresponds to the maximum porosity in the blocked cells, a satisfactory level of convergence was achieved with the highest residual in the region of 10^{-4}. The obtained geometry is shown in figure 4.5.1. The new geometry created in the box resembles the shape of a circular duct, but with a varying radius. The rough edges seen in the new geometry are the individual cells that have been blocked.

The flow changed significantly between the unmodified and adjoint optimised design space. Figure 4.5.2 shows streamlines for both the modified and unmodified geometry. The optimised geometry removes all of the large scale swirl and this should lower the pressure drop. However, the total pressure drop from inlet to outlet was about the same for both geometries. Since the solver is supposed to optimise with respect to lowering pressure loss, this was unexpected. One reason for the pressure drop not being lower in the modified design could be because of the rough walls created by the blocked cells. The mesh used here had approximately 300000 cells. To try and lower the effect of the rough edges a refined mesh with 800000 cells was tested as

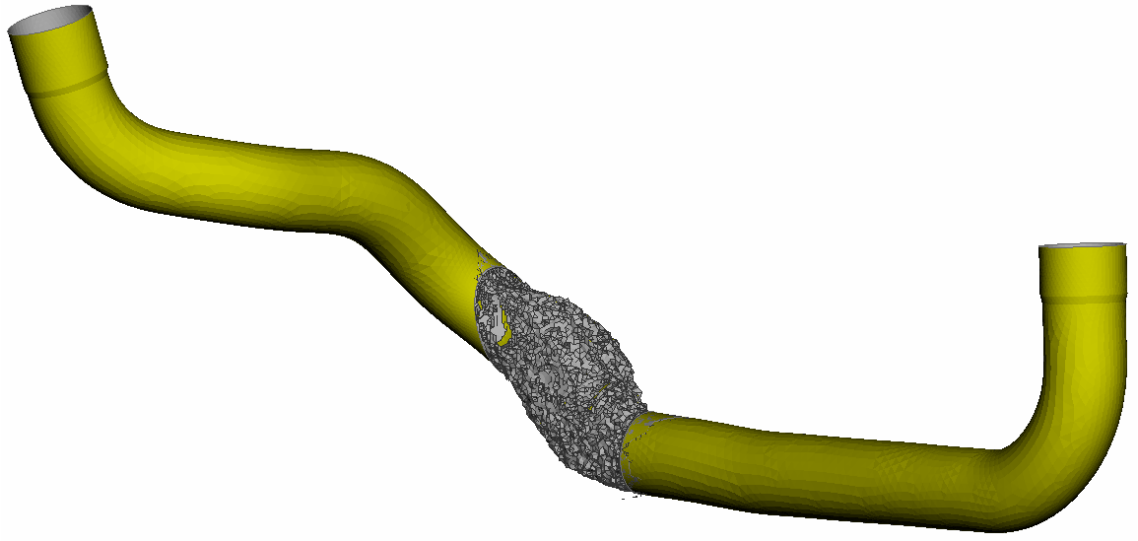


Figure 4.5.1: *Geometry obtained after adjoint optimisation*

well. The results were still the same and it was concluded that if the rough cell edges were responsible for the pressure drop being unchanged before and after optimisation, refining the mesh does not help. To get the real benefit of an adjoint optimisation with blocked cells the rough surface obtained would probably have to be imported into the CAD model and used as a guide when updating the design.

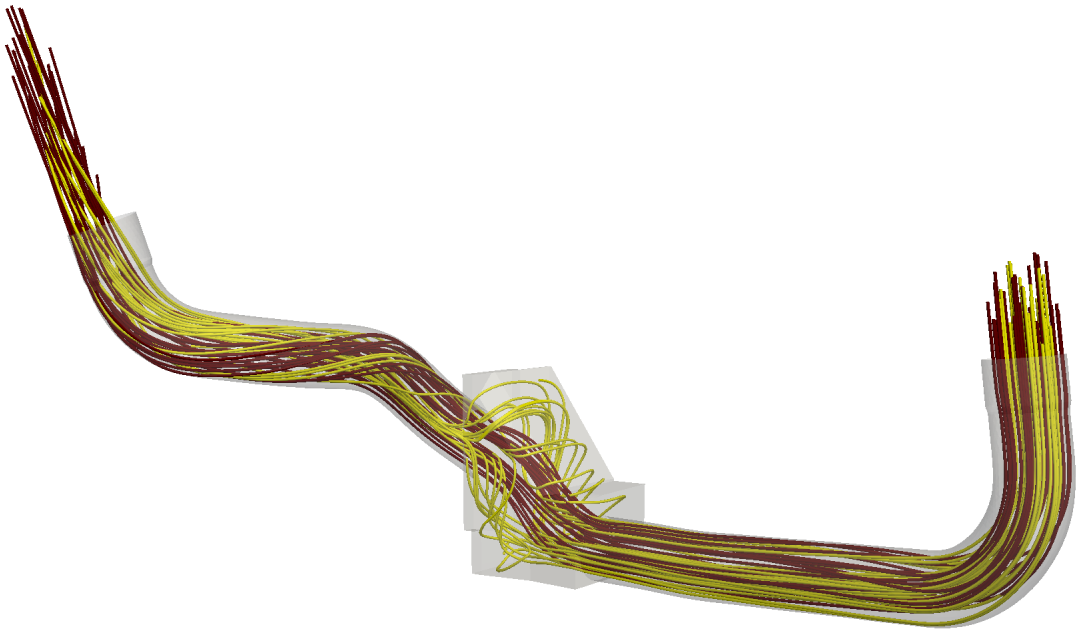


Figure 4.5.2: *Streamlines for original and optimised geometries*

5 Discussion and conclusions

The most important part of this work has been to assess whether OpenFOAM can be used as a tool to reliably predict the flow in automotive air induction systems. Also part of the thesis has been to investigate the possibilities and suitability of using OpenFOAM as a tool in optimising the shape of future air induction systems.

5.1 System simulations

Comparing the results from Fluent and OpenFOAM simulations it is evident that there are some differences between the softwares. The absolute numbers on pressure loss obtained with OpenFOAM were further from the experimental values than the ones obtained with Fluent. OpenFOAM simulations show a pressure drop almost 10% lower than the simulation from Fluent. The problem here is that already Fluent tends to under predict the losses compared to experimental results. There are also differences in where the losses take place in the system. Fluent shows a higher loss in the ducts compared to OpenFOAM, while the latter shows a higher loss in the intake. It is hard to tell exactly where these differences come from, but it should be noted that the same numerical settings and boundary conditions have not been possible to use. OpenFOAM couldn't handle the flow being prescribed at the outlet, but instead it had to be set at the inlets. If the same boundary conditions would have been possible to use in experiments and simulations they could have been validated in more thorough way against each other. The simulations involving porous zones turned out to be highly unstable. Despite several initiation techniques such as starting with incompressible flow, 1st order numerical schemes and low flow speeds the OpenFOAM simulations couldn't be made to use all second order schemes for the numerics. This will of course affect the results, but at least the most important field (momentum) could be run with a second order scheme. The tolerances of the linear solvers were set to low values. This increased solution time but was necessary to achieve reasonably low residuals. Even with the low tolerances, OpenFOAM had a speed advantage compared to Fluent.

The work carried out when writing this thesis has included testing a great number of combinations of URFs, numerical schemes and solver settings. Even so, it can not be ruled out that further tests and trials could find settings that can be used to better predict the air induction system flows. There is also the problem encountered with the oscillating flow speeds in the transition to a porous zone (the air filter). As no solution to this is presently known this must be considered a drawback for OpenFOAM when simulating complete air filter systems. Even if this phenomenon does not affect the pressure loss over the whole system in a significant way, it makes the uniformity index at the filter inlet useless. When simulating single ducts without an air filter there are no problems in using all second order numerical schemes, further indicating that it is the porous media implementation that suffers from a problem. In post-processing, the total pressure drop and uniformity indexes can easily be read in the log-file with the use of swak4foam. This is an advantage compared to Fluent as no post-processing is needed (or rather post-processing is done on-the-fly). OpenFOAM data can easily be converted to most post-processing software formats. It should be noted though that the conversion isn't optimal

as data from internal surfaces doesn't follow. The only way found to present plots of the data on internal surfaces such as figure 4.2.3 is to use the paraFoam wrapper for Paraview. One of the greatest advantages with OpenFOAM is that it is command-line based. This makes it easy to automatise and integrate with other softwares. Outside of the scope of this thesis, but nevertheless also significant merit for OpenFOAM is the possibility for users to add functionality.

The parallel scalability with regard to number of cores used does depend on certain variables such as the geometry of the mesh, the decomposition method used and resulting number of shared interfaces between processors. The main reason for the scalability not being linear is usually that the processors and nodes have to shuffle a lot of data between each other and the memory (depending on the number of shared faces). The speed of the interfaces between nodes and processors are one of the most important variables when considering the effectiveness in increasing the number of cores to use for a simulation. The test conducted reflects the optimal with the current setup used at VCC.

This thesis has shown that there are problems in simulating the air filter systems with OpenFOAM, the most severe being the oscillating velocities at the porous interface (see section 4.2.3). If a solution to this is found there are no substantial obstacles for using OpenFOAM as the main CFD-package for the air induction systems.

5.2 Topology Optimisation

When optimising the absolute numbers are not as important as long as the differences with changing geometry are correct. OpenFOAM is command-line based and this makes it very easy to integrate with optimisation softwares such as modeFrontier. Together with the fact that it is free [5] and thus lowering the licensing costs for simulation runs this makes OpenFOAM very suitable to use for DOE-based optimisation loops. The integrated pre- and post-processing such as the swak4foam script used in this thesis is also very useful in an optimisation loop as it enables extraction of a wide variety of data without user interaction.

The adjoint solver currently distributed with OpenFOAM (version 2.1.0) is a very basic solver. The solver optimises against one variable, total pressure loss. It also changes the geometry by adding blockages to the cells that contribute to pressure loss as suggested by the adjoint equations. While this works well to get an idea of where the losses occur, the geometry achieved with the blocked cells is less than optimal in the sense that it creates very rough walls. It is possible to extract this surface and use it as input in CAD software when improving a design. A better approach for changing the geometry would probably be to use some kind of CAD parametrisation or mesh morphing to get an optimised model directly.

Another thing to take into consideration with the adjoint method is that it does not take other factors such as manufacturability and manufacturing cost into account when computing the optimal shape. This can create good geometries with respect to the optimised variable, but which are not possible to manufacture. The result will therefore always require at least some interpretation by a designer before it can be used. For this reason the results obtained from an adjoint method optimisation are best suited as a guide to where changes need to be made in a design rather than using the geometry directly. A conclusion regarding the adjoint method

for optimising air induction system topologies is that it is a good tool to use for a designer in determining where to make changes. But it is not a tool producing a geometry ready for production.

6 Recommendations for further work

- Even though time consuming, continued investigations into finding numerical schemes etc. could possibly improve the performance and stability when using OpenFOAM for simulating complete air filter systems.
- Finding a solution to the porous boundary problem would enable OpenFOAM to be used in predicting filter utilisation, something which is now not possible.
- While the adjoint method holds great potential the solver currently distributed with OpenFOAM is of a very simple nature. To improve the usability of the adjoint method a new solver would have to be developed. On the wish list for a new solver would be, for example, the ability to handle compressible flows, more variables to optimise against (for example uniformity index) and to only apply the adjoint equations to certain parts of the mesh, for example a *cellZone*.
- Investigate the OpenFOAM meshing tool *snappyHexMesh* for meshing the air induction systems.

References

[1] F. M. White. *Fluid Mechanics*. Sixth Edition. McGraw-Hill, 2008.

[2] A. N. Kolmogorov. "Equation of turbulent motion of an incompressible fluid." In: *Physics* 6 (1941), pp. 56–58.

[3] W. P. Jones and B. E. Launder. "The prediction of laminarization with a two-equation model of turbulence." In: *International Journal of Heat and Mass Transfer* 15 (1972), pp. 301–314.

[4] H. K. Versteeg and W Malalasekera. *An Introduction to Computational Fluid Dynamics, The Finite Volmue Method*. Second Edition. Pearson Education, 2007.

[5] *GNU General Public License*. Version 3. 2007. URL: `http://www.gnu.org/licenses/gpl.html`.

[6] *OpenFOAM Userguide*. Version 2.1. OpenCFD, 2011. URL: `http://www.openfoam.org/docs/user/`.

[7] H. E. Hafsteinsson. "Porous Media in OpenFOAM". 2009.

[8] M. Dybeck. "CFD-procedures for Air Induction Systems". MA thesis. Luleå University of Technology, 2011.

[9] R. Schneider. "Applications of the Discrete Adjoint Method in Computational Fluid Dynamics". PhD thesis. University of Leeds, 2006.

[10] C. Hinterberger and M. Olesen. "Industrial application of continuous adjoint flow solvers for the optimization of automotive exhaust systems". In: ECCOMAS. 2011.

[11] *0000134: Unphysical velocity patterns at the interface of a porous zone*. 2011. URL: `http://www.openfoam.org/mantisbt/view.php?id=134`.

A Introduction to OpenFOAM

OpenFOAM is an abbreviation which stands for "Open source Field Operation And Manipulation" and is an open source numerical simulation software with extensive capabilities in solving fluid flows and other multi-physics problems. From the beginning (circa 1993) the software was called simply FOAM and was first developed as part of a PhD project at Imperial College London. In 2004 it became open source under the GNU GPL license [5] and changed name to OpenFOAM. Today it is developed by SGI® and widely used by both academic institutions and corporations worldwide.

The OpenFOAM distribution includes a lot of things, but the basis is the C++ libraries. The libraries contain the OpenFOAM classes which are designed to make it easier to write applications that solve continuum mechanics problems. Even though you still need some programming knowledge, a thorough knowledge of C++ is not a hard requirement to create or edit OpenFOAM applications. The classes provide a syntax which resembles the original equation. This example from the OpenFOAM userguide [6] is probably one of the simpler ones but shows the idea behind the OpenFOAM classes. The equation A.0.1

$$\frac{\partial \rho \mathbf{U}}{\partial t} + \nabla \bullet \phi \mathbf{U} - \nabla \bullet \mu \nabla \mathbf{U} = -\nabla p \tag{A.0.1}$$

is in OpenFOAM represented by the code:

```
solve
(
    fvm::ddt(rho, U)
  + fvm::div(phi, U)
  - fvm::laplacian(mu, U)
    ==
  - fvc::grad(p)
);
```

The OpenFOAM distribution includes a large number of solvers. Each solver is designed to solve a specific type of problem. Most are designed to solve some kind of CFD problem but there are also solvers for other types of problems such as electro magnetic and financial problems. The other type of application distributed with OpenFOAM are the utilities. They handle mostly pre- and post-processing functions such as mesh conversion and data manipulation.

Table A.0.1 and Table A.0.2 contains a selection of solvers and utilities that are part of the OpenFOAM distribution and have been used in this thesis.

Table A.0.1: A selection of solvers distributed with OpenFOAM used in this thesis

Solver	Description
simpleFoam	Incompressible steady-state solver utilising the SIMPLE pressure-velocity coupling
porousSimpleFoam	Same as the above but with included implicit or explicit porosity treatment.
rhoPorousMRFSimpleFoam	Further extension of the above solvers for handling compressible flow (i.e. variable density, hence the 'rho' in the name).

Table A.0.2: A selection of utilities distributed with OpenFOAM used in this thesis

Utility	Description
checkMesh	Checks the mesh for certain quality parameters such as skewness.
fluent3DMeshToFoam	Converts a mesh from fluent to OpenFOAM format.
foamMeshToFluent	Converts a mesh from OpenFOAM to fluent format.
foamDataToFluent	Converts data from OpenFOAM to fluent format.
ptot	Calculates the total pressure from pressure and velocity fields (if compressible also rho).
foamLog	Extracts residuals from a log file for plotting.
decomposePar	Decomposes a mesh together with data fields for parallel processing.
reconstructPar	Reconstructs the mesh and fields after parallel processing is done.
foamToVTK	Converts mesh and data to VTK format. Useful for example if Paraview is to be used on another computer.

A.1 Case setup

An OpenFOAM case is contained in a folder with the name of the case. Inside this folder the case has a basic file structure shown in Figure A.1.1. The *system/* folder holds files for setting options associated with the solution procedure. In the *constant/* directory the case is described, that is, it contains the mesh (subdirectory *polyMesh/*) and physical properties. The time directories contains the data for all the fields at different times. The "times" refers to actual time in the case of transient simulations, but in the case of steady state simulations it is only a means of keeping track of iterations. The *0/* directory contains initial values (the "0:th" iteration) and subsequent time directories are created during the simulation as specified in *controlDict*.

This file structure is almost the same for all OpenFOAM cases with small differences due to the required files for different solvers. The configuration files for solvers and utilities are called dictionaries. Table A.1.1 contains a description of the most important ones.

A summary of the necessary steps for setting up an OpenFOAM case are:

- Create the mesh for example with the *blockMesh* utility or with ANSA (Subsection 3.1.3)

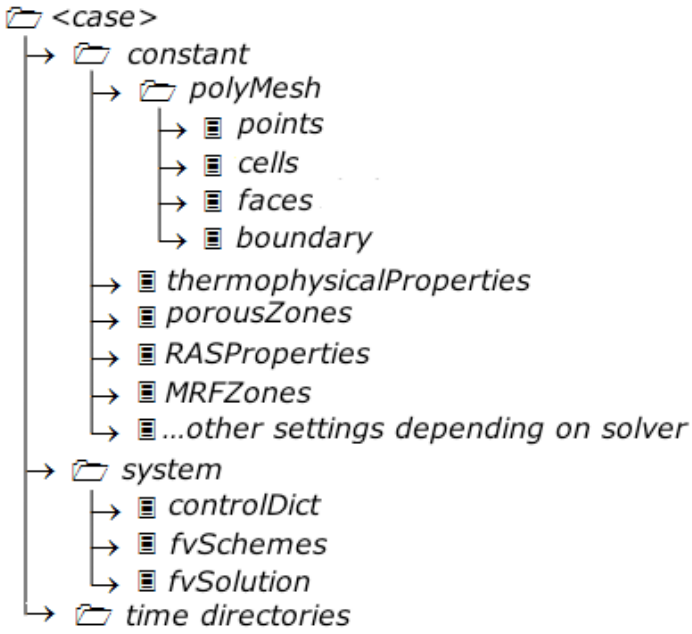


Figure A.1.1: *Directory structure of an OpenFOAM case*

Table A.1.1: Description of the most important OpenFOAM dictionaries used in this thesis

Dictionary	**Description**
controlDict	Controls start/stop, iterations, data writing and also contains function objects. Function objects are pieces of code that runs on every iteration.
fvSolution	Specifies the types of linear solvers, algorithms and under relaxation factors to use.
fvSchemes	Specifies numerical schemes.
boundary	Sets boundary types.
transportProperties	Defines fluid properties for incompressible solvers.
thermophysicalProperties	Specifies thermophysical properties of a fluid when using the energy equation (with compressible solvers).
RASProperties	Specifies RAS turbulence model, cf. LES modelling.
decomposeParDict	Options for the mesh decomposition required for parallel computing.

or convert the mesh from another format with a utility such as *fluent3DMeshToFoam*.

- Set boundary types in the *polyMesh/boundary* dictionary. In this thesis *patch* has been used for inlets/outlets etc and *wall* for walls.
- Set boundary conditions in the *0/* directory. Depending on the method used to create the mesh, it may be necessary to create/edit the volume field dictionaries manually.
- Set turbulence model in *RASProperties*/*LESProperties* dictionary for turbulent simulations.
- Set fluid properties such as μ and c_p. For most solvers this is done in either *transport-*

Properties or *thermophysicalProperties*.

- Set solution algorithms in *fvSolution* dictionary.
- Set schemes in *fvSchemes* dictionary.
- Set simulation controls in the *controlDict* dictionary.
- Decompose the case with *decomposePar* if using more than one core for solving.

When all the settings are set to appropriate values, the solver of choice is started by running the solver from the case directory. For parallel processing, the process of starting the solver depends on the type of parallel processing/queuing system used.

B swak4foam script

Below are examples of the entries used to calculate total pressure and uniformity index on the interior sections, in OpenFOAM defined as faceZones. Last is the code used to display mass flows on the inlets and outlet.

Entry for calculating the total pressure on a faceZone (mass weighted average)

```
Total-pressure_sec9_outlet
{
        functionObjectLibs ("libsimpleSwakFunctionObjects.so");

        type     swakExpression;
        outputControl    outputTime;
        valueType        faceZone;
        zoneName         sec9_outlet;
        expression
        "sum((0.5*rho*pow(mag(U),2)+p)*(area()*rho))/
        (sum(area()*rho))";

        accumulations
        (
                        average
        );
        verbose true;
        autoInterpolate true;
        warnAutoInterpolate      false;
}
```

Entry for calculating the uniformity index on a faceZone (area weighted average)

```
UI_sec9_outlet
{
        type     swakExpression;
        outputControl    timeStep;
        valueType        faceZone;
        zoneName                 sec9_outlet;
        accumulations
        (
                max
        );
        expression
        "1-sum((sqrt(pow((average(mag(U))-mag(U)),2))/
        (2*sum(area())*average(mag(U))))*area())";

        verbose          true;
        autoInterpolate true;
```

```
        warnAutoInterpolate      false;
}
```

Entry for printing mass flows across the boundaries

```
massFlow
{
        functionObjectLibs ("libsimpleSwakFunctionObjects.so");
        type     patchMassFlow;
        verbose  true;
        patches
        (
                inlet1
                inlet2
                pressureoutlet
        );
        factor   1;
}
```

C Python script

Python script for automation of the solving process. It changes from incompressible to compressible simulation settings and switches divergence schemes. Uses the Python library PyFoam.

```
#!/usr/bin/python

import os,sys,shutil

from PyFoam.Applications.Decomposer import Decomposer
from PyFoam.Applications.Runner import Runner
from PyFoam.Applications.PlotRunner import PlotRunner
from PyFoam.Execution.BasicRunner import BasicRunner

case=sys.argv[1]
proc=int(sys.argv[2])           #number of cores
ite_incomp=int(sys.argv[3])     #number of iterations for incompressible

#Schemes to use for divergence (in order of use)
divSchemesV=['upwind','linearUpwindV'];
divSchemes=['upwind','linearUpwind'];

#solvers to use
solver_incomp="porousSimpleFoam"
solver_comp="rhoPorousMRFSimpleFoam"

# Setting application to porousSimpleFoam
print '\nSetting application to porousSimpleFoam'

infilename='%s/system/controlDict' %case
outfilename='%s/system/controlDictTemp' %case
ifile = open(infilename, 'r')
ofile = open(outfilename, 'w')
lines = ifile.readlines()

for line in lines:
        words = line.split()
        for word in words:
                index = words.index(word)
                if word == 'application':
                        words[index+1] = solver_incomp+';'
                        continue
        newline=' '.join(words)
        ofile.write('%s\n' % newline)
ifile.close()
```

```
ofile.close()
os.remove(infilename)
os.rename(outfilename,infilename)

#Decompose for given number of cores
Decomposer(args=["--progress","--method=scotch",case,str(proc)])

#Set the divScheme to Gauss upwind; (first order accurate)
#shutil.copy ('%s/system/fvSchemes'%case ,'%s/system/fvSchemes.old'%case)
#print '\nSetting divScheme to Gauss %s for Momentum, k, eps, h and pressure

#infilename='%s/system/fvSchemes' %case
#outfilename='%s/system/fvSchemesTemp' %case
#ifile = open(infilename, 'r')
#ofile = open(outfilename, 'w')
#lines = ifile.readlines()
#
#for line in lines:
#        words = line.split()
#        for word in words:
#                index = words.index(word)
#                if word == 'div(phi,U)':
#                        words[index+2] = divSchemesV[0]+';'
#                        continue
#                if word == 'div(phi,k)':
#                        words[index+2] = divSchemes[0]+';'
#                        continue
#                if word == 'div(phi,epsilon)':
#                        words[index+2] = divSchemes[0]+';'
#                        continue
#                if word == 'div(phi,h)':
#                        words[index+2] = divSchemes[0]+';'
#                        continue
#                if word == 'div(U,p)':
#                        words[index+2] = divSchemes[0]+';'
#                        continue
#        newline=' '.join(words)
#        ofile.write('%s\n' % newline)
#ifile.close()
#ofile.close()
#os.remove(infilename)
#os.rename(outfilename,infilename)

#Run solver with incompressible solver
print '\nRunning 500 iterations with %s' %solver_incomp
Runner(args=["--proc=%d"%proc,"--progress",solver_incomp,"-case",case])
```

```
#Reconstruct the case to be able to manipulate fields
Runner(args=["reconstructPar","-case",case])

#change to compressible settings
#(p and phi dimensions, controldict, k & epsilon wall functions)
print '\nSwitching to compressible solver settings'

#k wallF
infilename='%s/500/k' %case
outfilename='%s/500/kTemp' %case
ifile = open(infilename, 'r')
ofile = open(outfilename, 'w')
lines = ifile.readlines()

for line in lines:
        line = line.replace("kqRWallFunction","compressible::kqRWallFunction
        ofile.write(line)
ifile.close()
ofile.close()
os.remove(infilename)
os.rename(outfilename,infilename)

#eps wallF
infilename='%s/500/epsilon' %case
outfilename='%s/500/epsilonTemp' %case
ifile = open(infilename, 'r')
ofile = open(outfilename, 'w')
lines = ifile.readlines()

for line in lines:
        line = line.replace("epsilonWallFunction","compressible::epsilonWallF
        ofile.write(line)
ifile.close()
ofile.close()
os.remove(infilename)
os.rename(outfilename,infilename)

#eps inlet
infilename='%s/500/epsilon' %case
outfilename='%s/500/epsilonTemp' %case
ifile = open(infilename, 'r')
ofile = open(outfilename, 'w')
lines = ifile.readlines()

for line in lines:
```

```
        line = line.replace("turbulentMixingLengthDissipationRateInlet","com
        ofile.write(line)
ifile.close()
ofile.close()
os.remove(infilename)
os.rename(outfilename,infilename)

#change application in controlDict
infilename='%s/system/controlDict' %case
outfilename='%s/system/controlDictTemp' %case
ifile = open(infilename, 'r')
ofile = open(outfilename, 'w')
lines = ifile.readlines()

for line in lines:
        line = line.replace(solver_incomp,solver_comp)
        ofile.write(line)
ifile.close()
ofile.close()
os.remove(infilename)
os.rename(outfilename,infilename)

#changing dimension for p to compressible dim
infilename='%s/500/p' %case
outfilename='%s/500/pTemp' %case
ifile = open(infilename, 'r')
ofile = open(outfilename, 'w')
lines = ifile.readlines()

for line in lines:
        line = line.replace("[0 2 -2 0 0 0 0]","[1 -1 -2 0 0 0 0]")
        ofile.write(line)
ifile.close()
ofile.close()
os.remove(infilename)
os.rename(outfilename,infilename)

#changing dimension for phi to compressible dim
infilename='%s/500/phi' %case
outfilename='%s/500/phiTemp' %case
ifile = open(infilename, 'r')
ofile = open(outfilename, 'w')
lines = ifile.readlines()

for line in lines:
        line = line.replace("[0 3 -1 0 0 0 0]","[1 0 -1 0 0 0 0]")
```

```
        ofile.write(line)
ifile.close()
ofile.close()
os.remove(infilename)
os.rename(outfilename,infilename)

#copy bc files for compressible to new result directory
shutil.copy ('%s/0/alphat'%case,'%s/500/alphat'%case)
shutil.copy ('%s/0/mut'%case,'%s/500/mut'%case)
shutil.copy ('%s/0/T'%case ,'%s/500/T'%case)

#Decompose the case again with new boundary conditions
Runner(args=["decomposePar","-force","-case",case])

#Run solver
print '\nRunning 500 iterations with %s' %solver_comp
Runner(args=["--proc=%d"%proc,"--progress",solver_comp,"-case",case])

#Change the divScheme to Gauss linear; (second order accurate) for Mom, k and
print '\nSetting divScheme to Gauss %s for momentum, k and eps' % '' .join(m

infilename='%s/system/fvSchemes' %case
outfilename='%s/system/fvSchemesTemp' %case
ifile = open(infilename, 'r')
ofile = open(outfilename, 'w')
lines = ifile.readlines()

for line in lines:
        words = line.split()
        for word in words:
                index = words.index(word)
                if word == 'div(phi,U)':
                        words[index+2] = divSchemesV[1]+' limited;'
                        continue
                if word == 'div(phi,k)':
                        words[index+2] = divSchemes[1]+' limited;'
                        continue
                if word == 'div(phi,K)':
                        words[index+2] = divSchemes[1]+' limited;'
                        continue
                if word == 'div(phi,epsilon)':
                        words[index+2] = divSchemes[1]+' limited;'
                        continue
        newline=' '.join(words)
        ofile.write('%s\n' % newline)
ifile.close()
```

```
ofile.close()
os.remove(infilename)
os.rename(outfilename,infilename)

#Run solver
print '\nRunning 500 iterations with %s' %solver_comp
Runner(args=["--proc=%d"%proc,"--progress",solver_comp,"-case",case])

# change the divScheme to Gauss linear; (second order accurate) for p
print '\nSetting divScheme to Gauss %s for pressure' % '' .join(map(str, divS

infilename='%s/system/fvSchemes' %case
outfilename='%s/system/fvSchemesTemp' %case
ifile = open(infilename, 'r')
ofile = open(outfilename, 'w')
lines = ifile.readlines()

for line in lines:
        words = line.split()
        for word in words:
                index = words.index(word)
                if word == 'div(U,p)':
                        words[index+2] = divSchemes[1]+' limited;'
                        continue
        newline=' '.join(words)
        ofile.write('%s\n' % newline)
ifile.close()
ofile.close()
os.remove(infilename)
os.rename(outfilename,infilename)

#Run solver
print '\nRunning 500 iterations with %s' %solver_comp
Runner(args=["--proc=%d"%proc,"--progress",solver_comp,"-case",case])

# change the divScheme to Gauss linear; (second order accurate) for h
print '\nSetting divScheme to Gauss %s for enthalpy' % '' .join(map(str, divS

infilename='%s/system/fvSchemes' %case
outfilename='%s/system/fvSchemesTemp' %case
ifile = open(infilename, 'r')
ofile = open(outfilename, 'w')
lines = ifile.readlines()

for line in lines:
        words = line.split()
```

```
        for word in words:
                index = words.index(word)
                if word == 'div(phi,h)':
                        words[index+2] = divSchemes[1]+' limited;'
                        continue
        newline=' '.join(words)
        ofile.write('%s\n' % newline)
ifile.close()
ofile.close()
os.remove(infilename)
os.rename(outfilename,infilename)

#Run solver
print '\nRunning 500 iterations with %s' %solver_comp
Runner(args=["--proc=%d"%proc,"--progress",solver_comp,"-case",case])

#Reconstruct the case
Runner(args=["reconstructPar","-case",case])
}
```