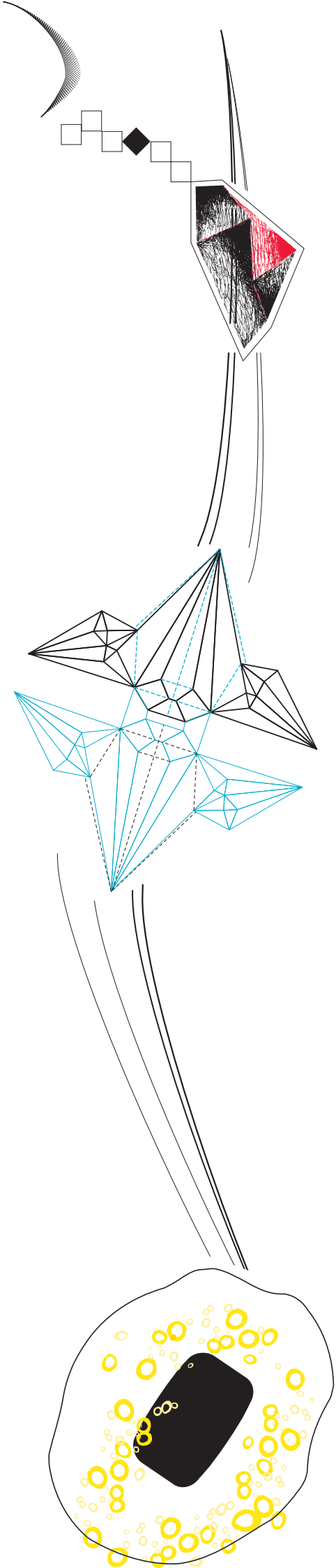


Internship Report
CIRA

Lennart van der Windt (s1005251)

Italian Aerospace Research Centre

UNIVERSITY OF TWENTE.

Contents

1 Introduction

At the Italian Aerospace Research Centre (CIRA), research is done on many interesting topics regarding aviation and aerospace applications. One of these applications is in-flight icing, where, at certain conditions, ice accumulates on all front facing surfaces during flight. The added weight is minimal and thus insignificant [2], but the ice can cause sensors and such to malfunction and can even influence the aerodynamic properties through a change in the aerodynamic shape of, for example, the wings. This can cause the aircraft to lose lift up to the point that flight can no longer be sustained. Numerous accounts of airplane crashes due to icing have been reported in the last couple of decades, calling for an immediate solution to the icing problem. Several anti- and de-icing systems were developed, but most of these require design compromises because of weight, applicability and power consumption issues. It is therefore still important to gain a better understanding of why and how ice accretion on aircraft takes place. In this work, an attempt is made to use CFD (computational fluid dynamics) software to make a full icing simulation.

1.1 In-Flight Icing

To be able to perform icing analyses using CFD software, the mechanics of icing need to be known and understood. Icing can occur at temperatures between $-40C^\circ < T < 0^\circ C$ and in areas with a high liquid water content (LWC), e.g. in clouds. In these conditions, Supercooled Large Droplets (SLD) are present, which are liquid droplets with a temperature below the freezing point. The size of the droplets is also very important and is defined by the Mean Volume Diameter (MVD). The LWC and MVD together determine the cloud conditions. When these droplets hit a solid surface like an aircraft, they can solidify, depending on the exact conditions. The fraction of water that solidifies instantly when contact is made with the airfoil is called the freezing fraction, which is an important parameter in determining the type of ice which is formed.

Roughly speaking, two types of icing can be identified. The type depends greatly on the cloud conditions, ambient temperature and air speed of the aircraft. Firstly there is rime ice, which occurs when the droplets are relatively small and the temperatures are very low: $T < 15^\circ C$. In these conditions, most if not all of the droplets freeze immediately upon impact with no runback water i.e. this is characterised by a large freezing fraction. Because of this, the shape of rime ice roughly follows that of the airfoil, so that the aerodynamic losses are smaller as compared to the second type of ice: glaze ice [1]. Also, rime ice is relatively brittle, so that it can be combatted easily using de-icing or anti-icing systems.

Secondly, there is glaze (or clear) ice, which forms in milder conditions ($-15^\circ C < T < 0C^\circ$) and when the droplets are relatively large. In these conditions, the freezing fraction is relatively small, so that some of the water freezes on the surface, while some remains liquid and forms a liquid layer on top of the ice and/or airfoil surface. This runback water will then flow to other regions on the airfoil, subject to gravity and air shear forces as shown in figure 1.1 [6], where it can solidify later or not at all. This can cause the ice to build up in non-uniform shape, so that the aerodynamic properties of the airfoil can be greatly affected. Due to a small build-up of 0.4 mm of glaze ice on the leading edge of an airfoil, the lift of said airfoil can be reduced by as much as 25%, whilst the stall angle of attack can be reduced by as much as 6 degrees [4].

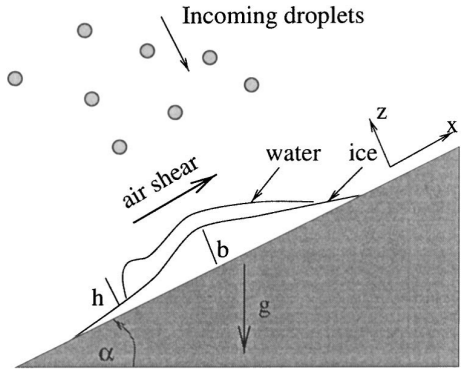


Figure 1.1: *The acting forces in a glaze icing problem*

The two types of ice are shown schematically in figure 1.2.

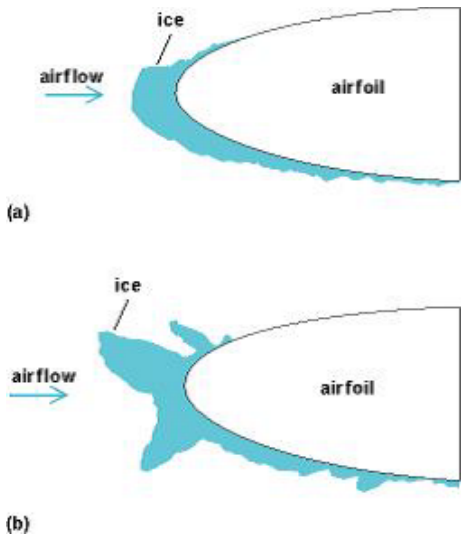


Figure 1.2: *Schematic representation of rime (a) and glaze (b) ice*

1.2 OpenFOAM

Open Source Field Operation and Manipulation, or *OpenFOAM* for short, is an open source CFD software package. It can be used for a wide variety of applications, including but not limited to engineering and scientific flow problems. In essence, *OpenFOAM* is a big C++ library which can create executables like solvers and utilities. Utilities are tools to perform tasks involving data manipulation. In terms of solvers, the standard applications supplied with *OpenFOAM* can be used, or user made applications can be implemented. This structure can be seen in figure 1.3 [5].

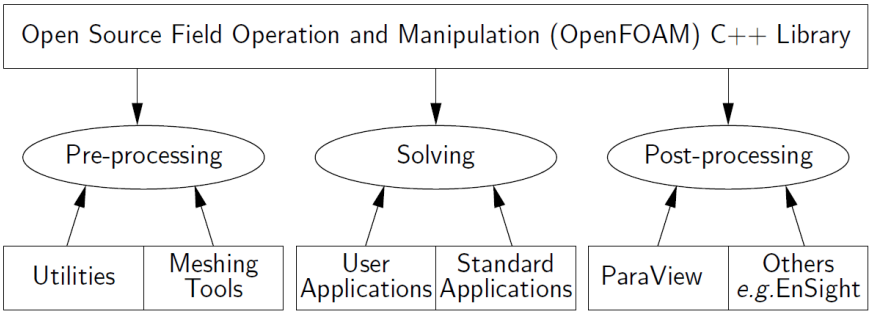


Figure 1.3: *Schematic representation of the OpenFOAM software structure*

In this work, OpenFOAM will be used to analyse standard two icing cases on a certain three dimennional wing. Both rime ice and glaze ice conditions will be investigated to test the performance of *OpenFOAM* of standard solutions to icing problems. These can then be used as a reference point for future analyses in *OpenFOAM* for icing problems, such as the implementation of de-icing or anti-icing equipment.

A glaze icing analysis in *OpenFOAM* consists of a number of steps, as shown in figure 1.4 [3]. In the first step, a mesh around the desired geometry is created. In OpenFOAM, all meshes are three-dimensional, but these can artificially be made two-dimensional by choosing a mesh thickness of 1 element in a certain direction and by applying so-called *empty* boundary conditions on the sides. A more detailed explanation of the meshing process will be given in chapter 2. Next, the aerodynamic solution is calculated with clean air at the given flight and ambient conditions. In the third step, droplet trajectories and impingement rates are calculated from the aerodynamic solution and the cloud conditions (which were defined by the LWC and MVD). This is done for a certain time span δt, so that enough droplets hit the airfoil to create a water film. In the meantime, this time span needs to be sufficiently small to describe the icing process discretely, whilst this is a continuous process in reality. During this step, no effects concerning the solidification or evaporation is taken into account. In the fourth step, the thermodynamic solution is calculated, in which the (partial) freezing of the water film is incorporated. Lastly, the aerodynamic shape in recalculated by adding the accreted ice to the airfoil. If the desired time span has not been reached at that point, the process is reinitialised at the meshing stage with the updated geometry.

OpenFOAM can save prescribed time solutions into directories, such that only the desired time steps are saved. For example, the directory *0* contains the initial conditions at $t = 0s$, while the directory *20* will contain the solution at $t = 20s$, without saving any solutions to the time steps in between 0 and 20 seconds. The solutions contained in these directories can then be viewed using *OpenFOAM*s post-processing tool *paraFOAM*, which contains the open source visualization tool *ParaView*.

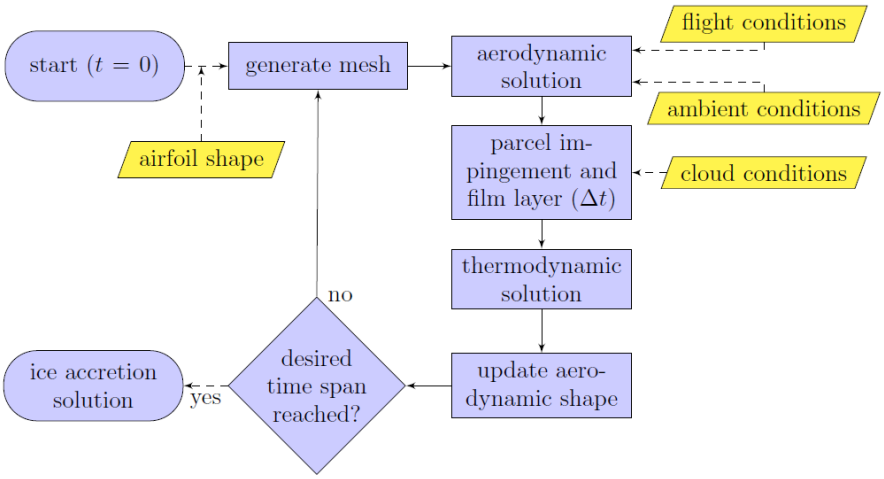


Figure 1.4: *Schematic representation of the computational steps in OpenFOAM*

1.3 Geometry and conditions

In this section, the geometrical outline of the used wing is given. All cross-sections are made using the GLC-305 airfoil, which can be seen in figure 1.5. In figure 1.6, the design rules of the wing are given, with which the geometrical triangulation is constructed. In figure 1.7, a top view of the wing is given. Do note that in these figures, all dimensions are given in inches.

This geometry is chosen, as numerous icing experiments were performed on the described wing by Vargas et al. [8] in 2002. They set up a scale model of the wing in an icing wind tunnel at NASA's Glenn Research Center and performed experiments with different conditions.

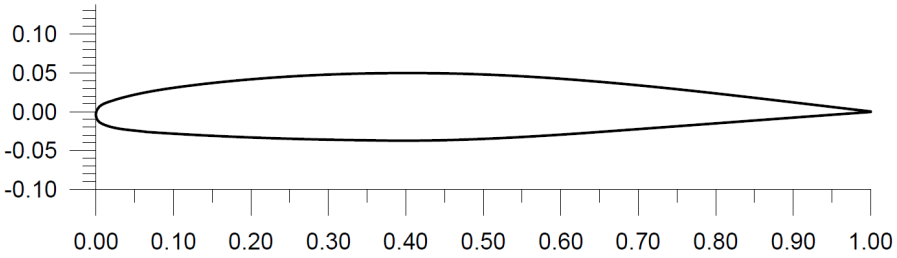


Figure 1.5: *Cross-section of the wing at any point*

WING PLANFORM

1. 5 ft semispan (72% of WSU tunnel height)
2. Root chord = 25.2 in, Tip chord = 10.08 in, MAC = 18.72 in
3. t/c varies linearly from root to tip
4. Planform Area (half wing, S/2) = 7.35 ft^2; wing area to test section area ratio = 7.35/(7x10) = 0.105
5. taper ratio = 0.4, trapezoidal planform, linear chord variation from root to tip
6. AR = $b^2/S = 10^2/(2*7.35) = 6.8027$
7. 25° quarter chord sweep, 28° leading edge sweep

WING GEOMETRIC TWIST AND DIHEDRAL

1. Geometric twist about 25% local chord (0° at root, -4° at tip)
2. Dihedral = 0°**AILERON CONTROL SURFACE**

1. 25% local chord (from 70% to 95% semispan); $C_{root} = 3.66$ in, $C_{tip} = 2.712$ in
2. Elliptical leading edge (pivot at 80% local wing chord), Gap = 0.5% local chord
3. Deflection range: -20° to +20°
4. Aileron area behind the hinge line, $S_a = 0.2647\ ft^2$
5. Aileron average chord behind the hinge line, $c_a = [(2.163+2.92)/2/12] = 0.2118$ ft

Figure 1.6: *Parameters of the wing*

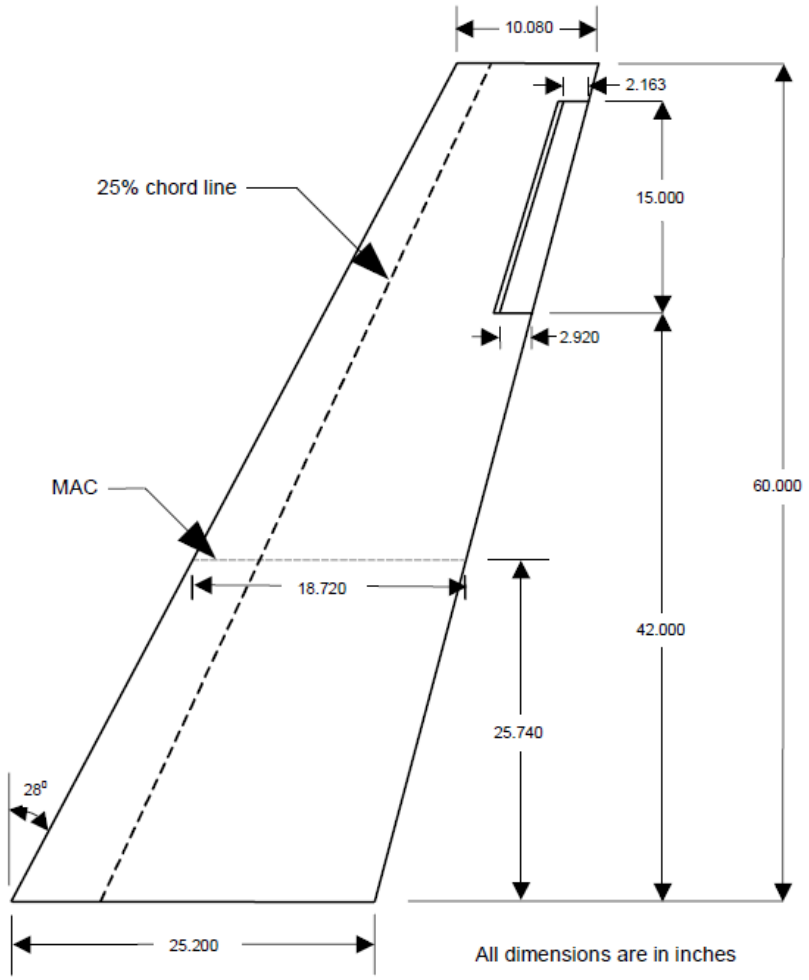


Figure 1.7: *Top view of the used wing*

Conditions

From the tested conditions used by Vargas et al. [8], two are chosen to model in *OpenFOAM*. The most important features are shown in table 1.1. Condition 1 equates to a glaze icing condition, whereas with condition 2, rime icing is expected.

Parameter	**Condition 1**	**Condition 2**
Angle of attack	4	6
Air velocity	111.11 m/s	90 m/s
LWC	0.68	0.51
MVD	20 μm	14.5 μm
Temperature	263 K	255 K

Table 1.1: *Test run data from Vargas et al. [8], as will be used in* OpenFOAM

1.4 Goal

OpenFOAM has been used extensively at CIRA for a variety of simulations, including icing analyses on various shapes. However, a full 3D wing has not yet been analysed in OpenFOAM at CIRA. The goal of this work is therefore to create a basis for an full icing analysis in OpenFOAM on a full 3D wing. The wing chosen is a GLC-305 wing with a span of 1.52 m, which is also swept, tapered and has a twist from $0°$ at the root to $-4°$ at the tip (see section 1.3).

To do this, three aspects of an icing investigation are carried out in OpenFOAM to see how they perform. First of all, the meshing of a full 3D wing is performed. Secondly, the aerodynamics around the airfoil are calculated. And lastly, the impingement of droplets onto the airfoil is investigated.

2 Meshing

One of the most important aspects of any CFD analysis is the meshing. Without a good mesh, the used solver might not converge, the geometry might be described badly and the obtained results will be poor. In this study, the built-in meshing software of OpenFOAM is used: *snappyHexMesh* (SHM).

2.1 snappyHexMesh

Making a mesh using *snappyHexMesh* roughly consists of three parts: making a background mesh, using the actual SHM utility and checking the mesh. SHM itself also consist of three parts: creating a castellated mesh, creating a snapped mesh and adding layers to the mesh. All steps will be explained below.
Creating a mesh always starts with the geometry to be described. This is usually done with an STL file or a similar format, see figure 2.1.

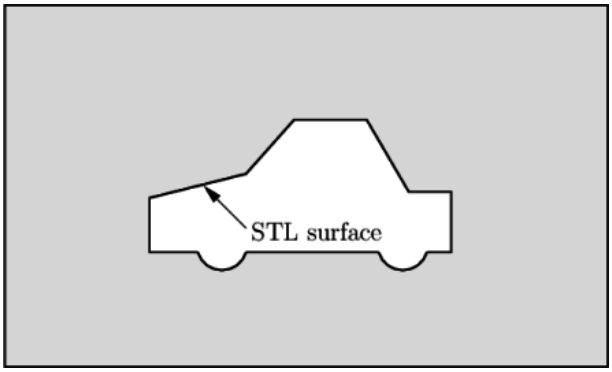


Figure 2.1: *STL surface as starting point*

From here, a background mesh is created, which is just a very coarse mesh disregarding the geometry and consists preferably of square or cubic elements. Every refinement level specified splits the cell edges in half, quadrupling the amount of cells. This is all shown in figure 2.2.

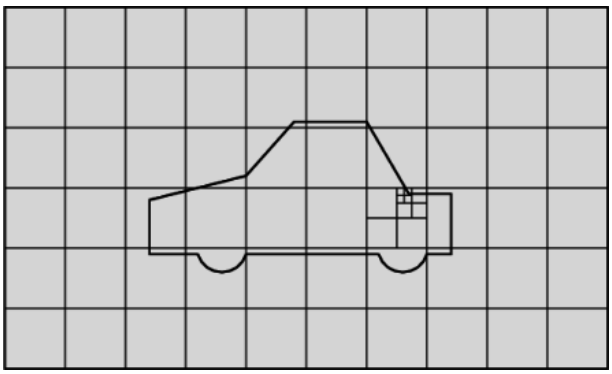


Figure 2.2: *Background mesh and refinement levels*

For each surface in the geometry (in the example mesh, only one), a level of refinement should be specified,

as well as the number of normal cell layers per level. This is to ensure that small cells don't appear too close to bigger cells, which can cause converging problems for some solvers. In the example, this is not shown. The result can be seen in figure 2.3.

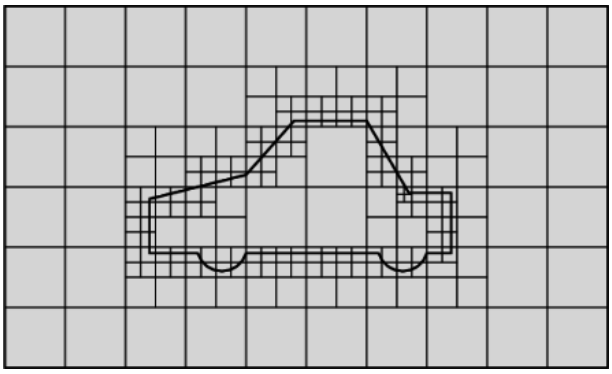


Figure 2.3: *Surface refinement*

To make sure that only the relevant flow area is meshed (either the inside or outside of the geometry), one of the areas is specified for deletion. Inside this area, all cells are deleted which have no intersection with the geometry. On top of this, specified areas are refined up to the indicated level. This is particularly useful for areas of interest which need a finer mesh, such as the wake of an airfoil. These areas can be specified as a variety of shapes, such as cubes, spheres and cylinders. The result of this step is shown in figure 2.4.

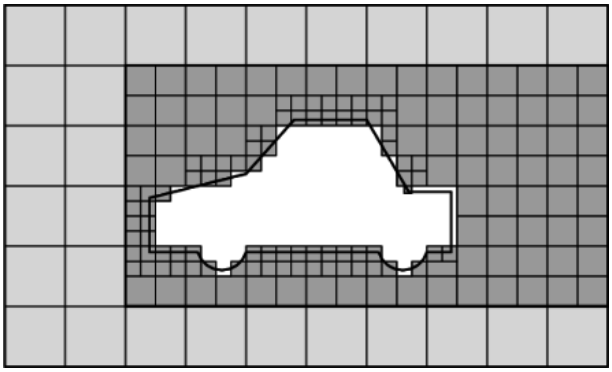


Figure 2.4: *Regional refinement and cell deletion*

The next step is where the utility gets its name from. The vertex points of the cells get 'snapped' onto the STL surface. With some relaxation sweeps, the surface is then smoothed and bad cells are fixed.

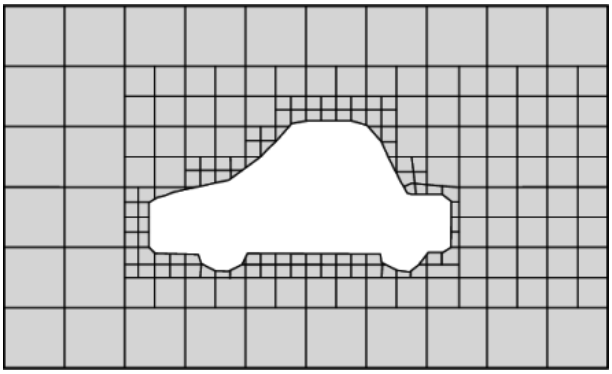


Figure 2.5: *Snapping of the mesh*

The last, optional step is to add layers to the surface. This can be done for the entire geometry, or only at parts of interest. The aim is to get a better resolution at these places to, for example, resolve a boundary layer better.

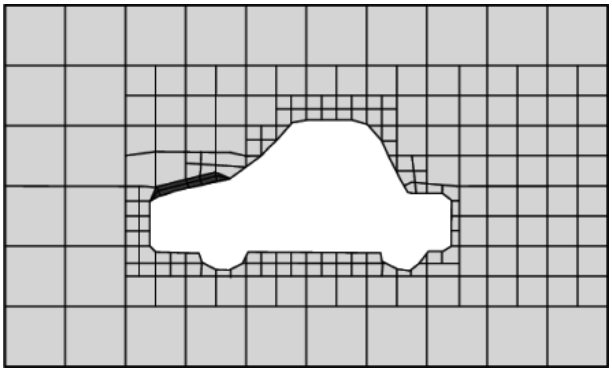


Figure 2.6: *Adding layers to the mesh*

2.2 2D Mesh

Firstly a 2D mesh is made for a section of the GLC-305 airfoil as a test. This way, any problems can be detected faster, as a 2D mesh takes much less time to generate as compared to a 3D mesh. The domain is ranging from 5 cord lengths in front to 15 cord lengths behind the airfoil, and 5 cord lengths both above and below the airfoil. The entire mesh is shown in figure 2.7, where it can also be seen that refinement boxes have been added when getting closer to the airfoil. This is done because near the airfoil, the biggest deviations from the far field flow conditions are expected, especially in the wake of the airfoil. Further away form the airfoil, little deviations from the far field conditions are expected, so a coarser mesh is used here.

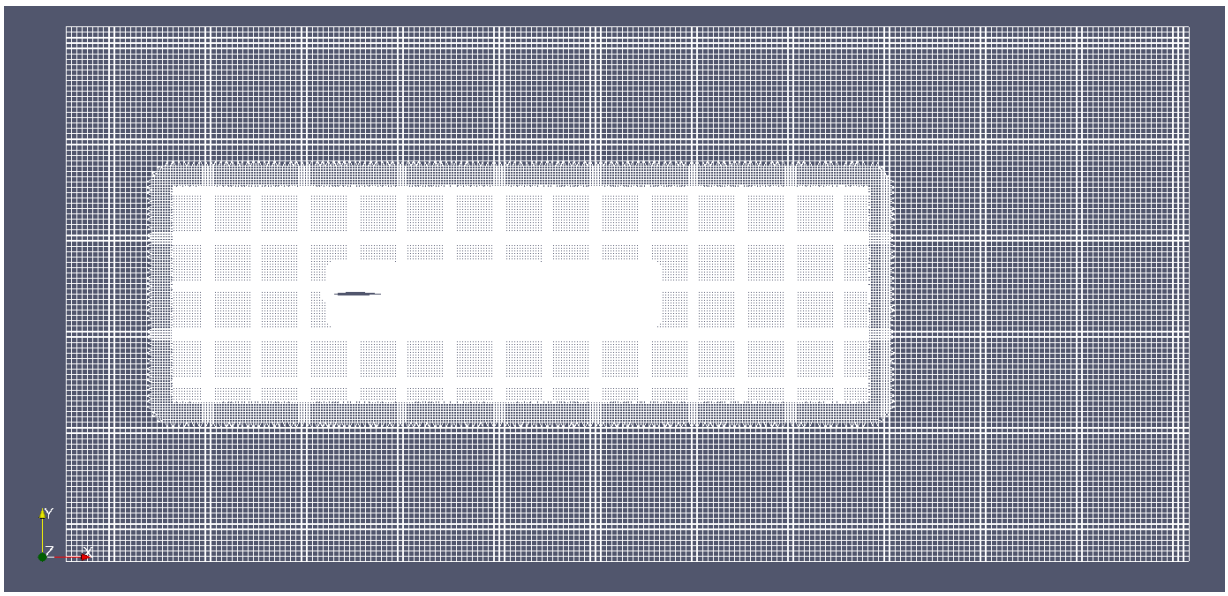


Figure 2.7: *Total 2D mesh*

Even closer to the airfoil, another refinement box has been added: a cylinder at the leading edge (remember that all meshes in OpenFOAM are 3D). This is done because both in terms of aerodynamics, impingement and icing, the leading edge is the most interesting area and therefore needs to be resolved well. A problem with the mesh arises on the top of the airfoil, a little past the mid-chord point. Here, SHM is somehow not able to place the layers on the surface and distorts the uniformity of the mesh locally. Other settings in the utility did not yield better results. The mesh close to the airfoil can be seen in figure

2.8 and the error in the mesh can be seen more clearly in figure 2.9.

The origin of the error is unknown, but it is hypothesized that this has to do with the quality of the used STL geometry. A close inspection of the STL surface revealed that there were very small local deformations in the surface. It was also at these locations, that the distortions in the mesh originated. Some of these distortions could be partially solved by altering the "featureAngle" setting in the layer options section of SHM. The value cannot just be increased or decreased to get better results, but every geometry will have a unique value. It was however, not possible to get rid of the mesh anomalies altogether. The best result is the mesh shown here.

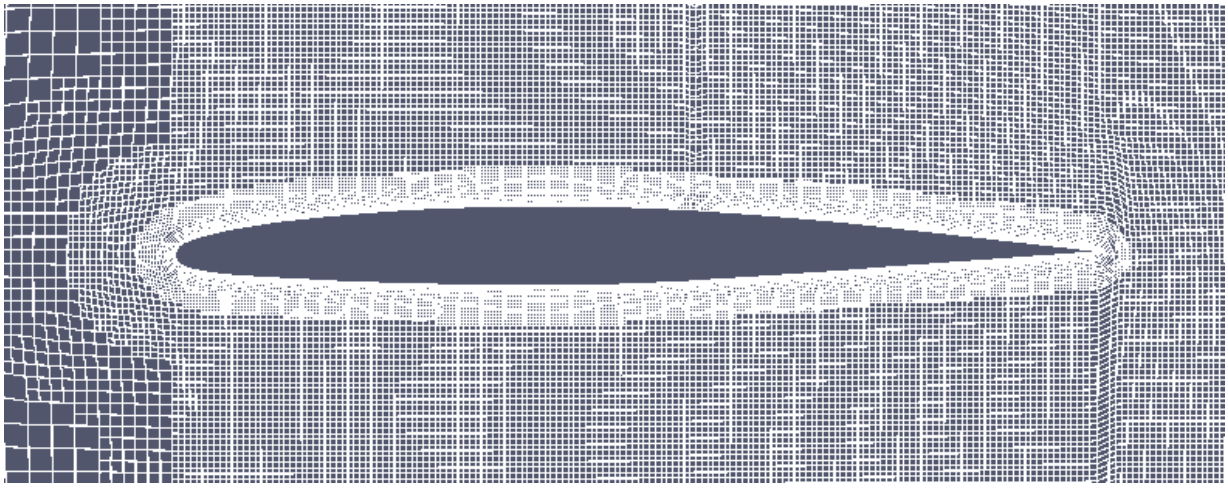


Figure 2.8: *Zoomed in 2D mesh*

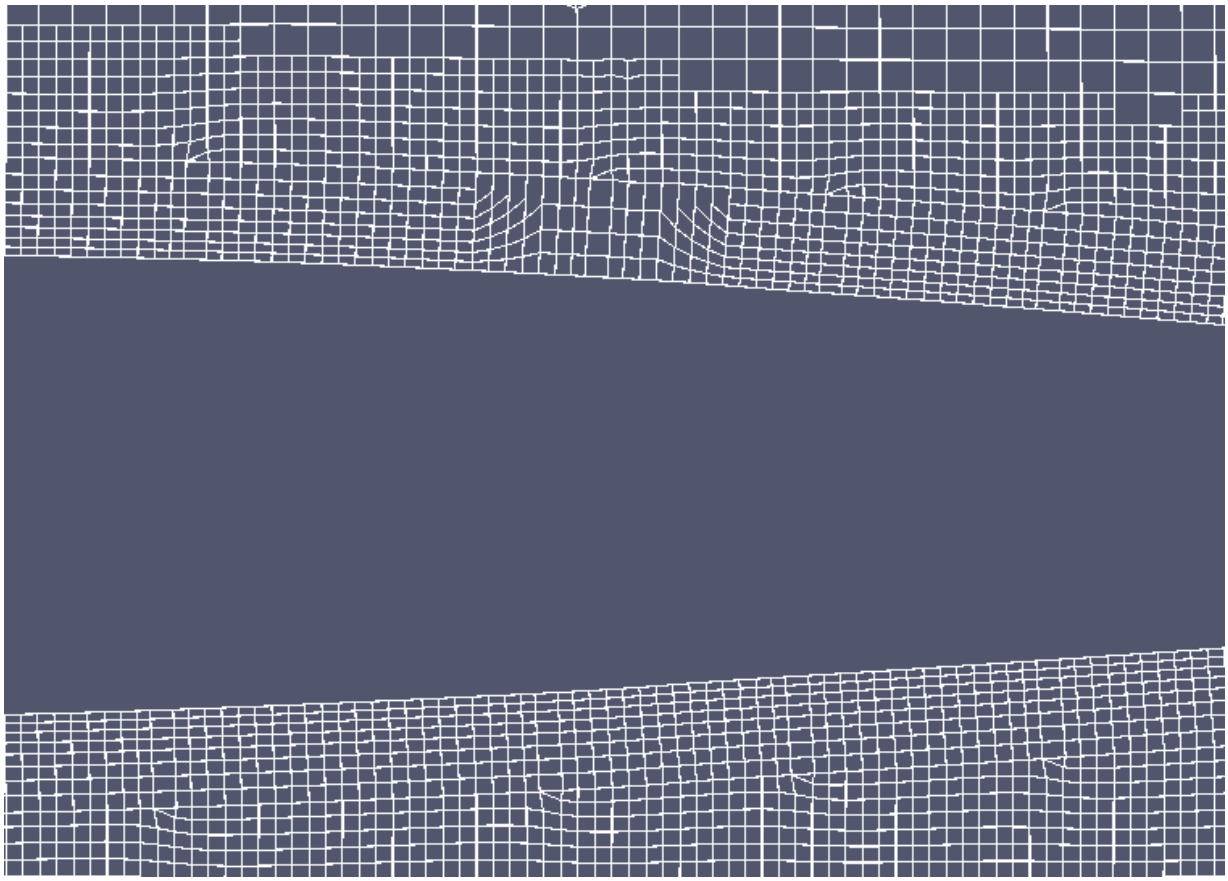


Figure 2.9: *Error in the mesh layers*

2.3 3D Mesh

The 3D mesh is basically constructed in the same way as the 2D mesh. However, fewer refinement iterations have been applied in order to restrict the amount of cells in the mesh. An overview of the mesh can be seen in figure 2.10, whilst a section of the mesh can be seen in figure 2.11.

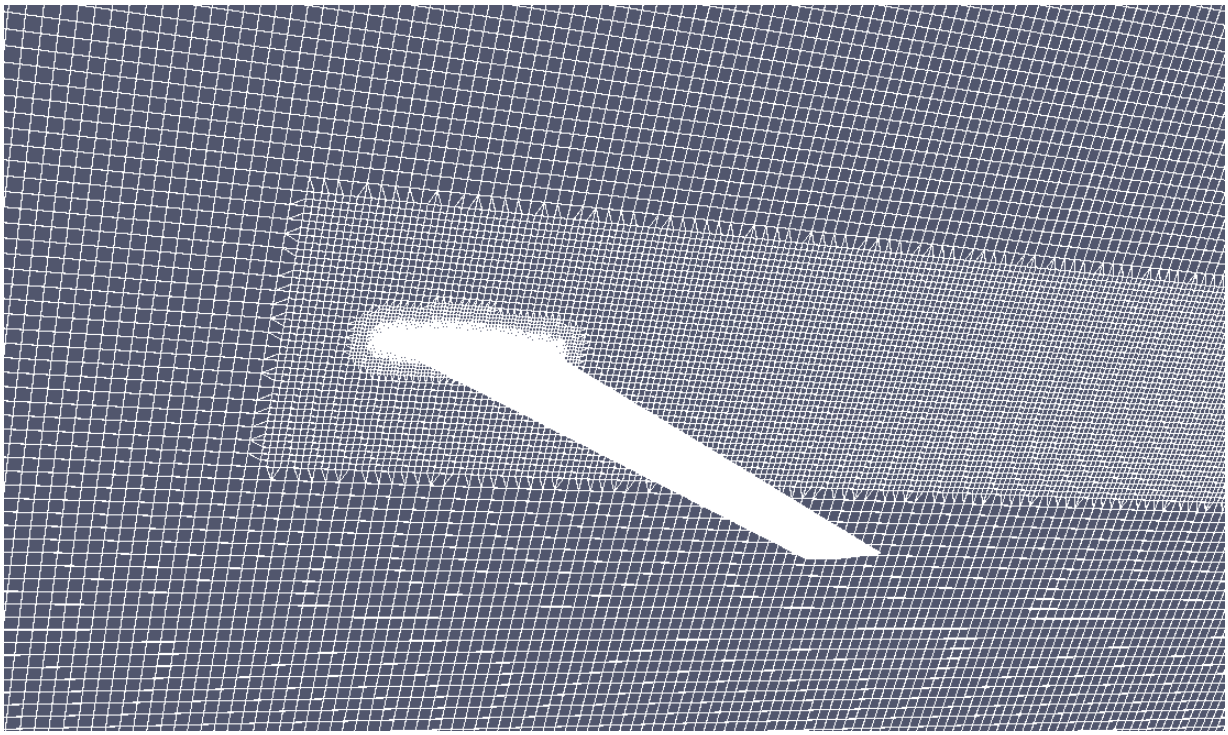


Figure 2.10: *View of the 3D mesh*

When zooming in on the wing, it can be seen that no layers have been added by SHM on the surface of the wing. The reason for this is still unknown, but none of numerous SHM configurations tried yielded any layers. It is again hypothesized that the quality of the STL file is the cause, but this is unconfirmed. It has to be noted that without surface layers, the boundary layer on the wing will probably not be predicted accurately.

In conclusion it can be stated that just a very rudimentary mesh has been obtained. As it turned out that very little could be done to improve the mesh further, this mesh will be used anyway in the remainder of this work.

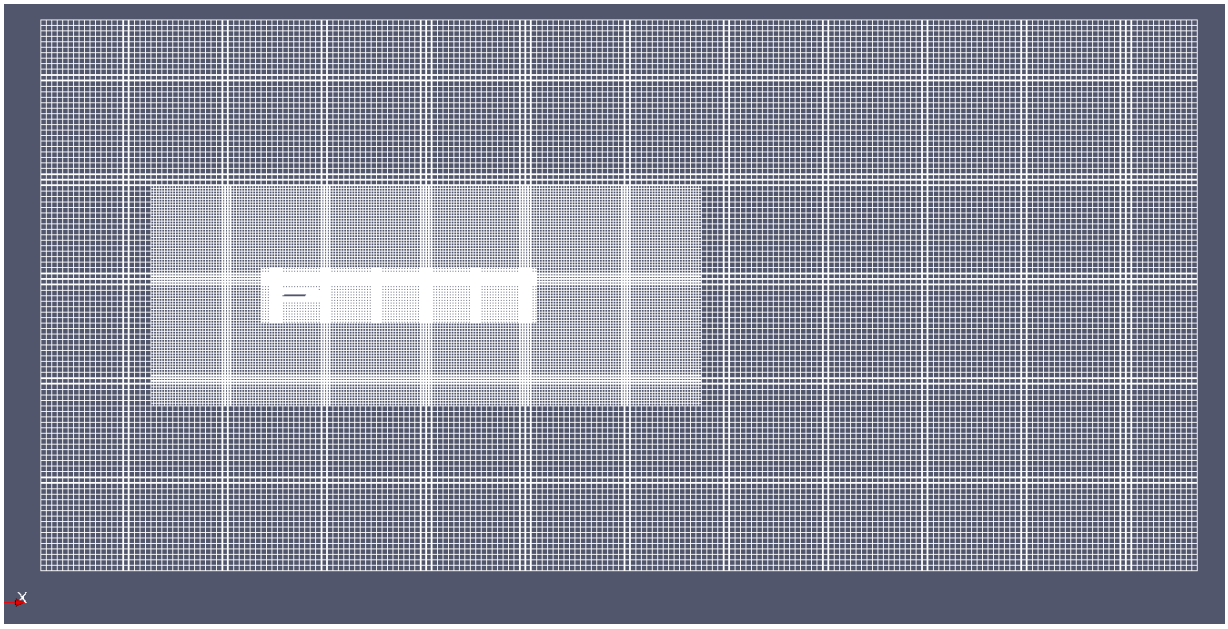


Figure 2.11: *Section of the 3D mesh*

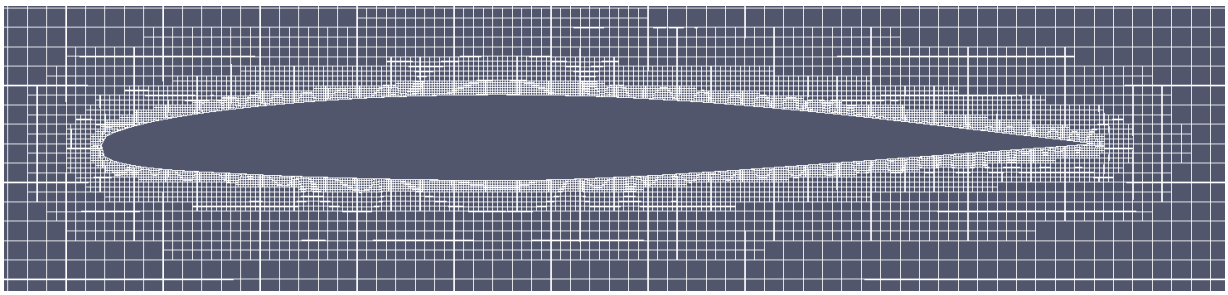


Figure 2.12: *Mesh near the wing surface*

3 Aerodynamics

After a mesh is generated, the first step in computing an icing analysis is to calculate the pressure and velocity field around the airfoil for a certain set of conditions. For the GLC-305 wing, no analytical or experimental solution is available, so the validity of the solution is hard to judge. Instead, pressure and velocity plots are visually inspected, as well as plot of the pressure coefficient around the airfoil. The conditions for which aerodynamic solutions are calculated are shown in table 3.1.

Parameter	**Condition 1**	**Condtion 2**
Free stream velocity	111.11 m/s	90 m/s
Angle of attack	4°	6°

Table 3.1: *Flow conditions for the aerodynamic simulations*

The aerodynamic solution around the airfoil is computed using *rhoPimpleFoam*, which is a transient solver with compressibility effects. This solver is chosen, as experience at CIRA has shown that using the steady state solver can sometimes yield unrealistic results. This can take a lot of time to compute, but the computational time can be reduced by imposing an initial solution. This solution should be a reasonable approximation of the real solution. To this end, *simpleFoam* is used, which is a stationary solver without compressibility effects. In this chapter, both *simpleFoam* and *rhoPimpleFoam* are described, after which the results are presented.

3.1 Incompressible solution - simpleFoam

An initial solution is made using the semi-implicit method for pressure-linked equations, or *SIMPLE* for short. The algorithm was described by OpenFOAM [7] and the general procedure is reprinted here. The basis of *simpleFoam* consists, naturally, of the Navier-Stokes equations:

$$\nabla \cdot \left(\rho \bar{U}\right) = 0 \tag{3.1}$$

$$\frac{\partial \bar{U}}{\partial t} + \nabla \cdot (\bar{v}\bar{v}) - \nabla \cdot (\nu \nabla \bar{v}) = -\nabla p \tag{3.2}$$

SimpleFoam assumes the density ρ and the kinematic viscosity ν to be constant, so this greatly simplifies the equations. However, there is still no straightforward solution, as no explicit equations for the pressure is available. This is solved in the following way.

The momentum equation is semi discretised in the following way:

$$a_p \bar{U}_p = H(\bar{U}) - \nabla p \tag{3.3}$$

where

$$H(\bar{U}) = -\sum_n a_n \bar{U}_n + \frac{\bar{U}^o}{\delta t} \tag{3.4}$$

The first term represents the matrix coefficients of the neighbouring cells multiplied by their velocity, while the second part contains the unsteady term and all the sources except the pressure gradient.

The continuity equation is discretised as:

$$\nabla \cdot \bar{U} = \sum_f \bar{S} \cdot \bar{U}_f = 0 \tag{3.5}$$

where $\bar{S}$ is outward-pointing face area vector and $\bar{U}_f$ the velocity on the face.

The velocity on the face is obtained by interpolating the semi discretised form of the momentum equation as follows:

$$\bar{U}_f = \left(\frac{H(\bar{U})}{a_p} \right)_f - \frac{(\nabla p)_f}{(a_p)_f} \tag{3.6}$$

By substituting this equation into the discretised continuity equation obtained above, pressure equation is obtained:

$$\nabla \cdot \left(\frac{1}{a_p} \nabla p \right) = \nabla \cdot \left(\frac{H(\bar{U})}{a_p} \right) = \sum_f \bar{S} \left(\frac{H(\bar{U})}{a_p} \right)_f \tag{3.7}$$

The algorithm uses the above equations in the following iterative procedure:

1. Set the boundary conditions.
2. Solve the discretized momentum equation to compute the intermediate velocity field.
3. Compute the mass fluxes at the cells faces.
4. Solve the pressure equation and apply under-relaxation.
5. Correct the mass fluxes at the cell faces.
6. Correct the velocities on the basis of the new pressure field.
7. Update the boundary conditions.
8. Repeat till convergence.

3.2 Compressible solution - rhoPimpleFoam

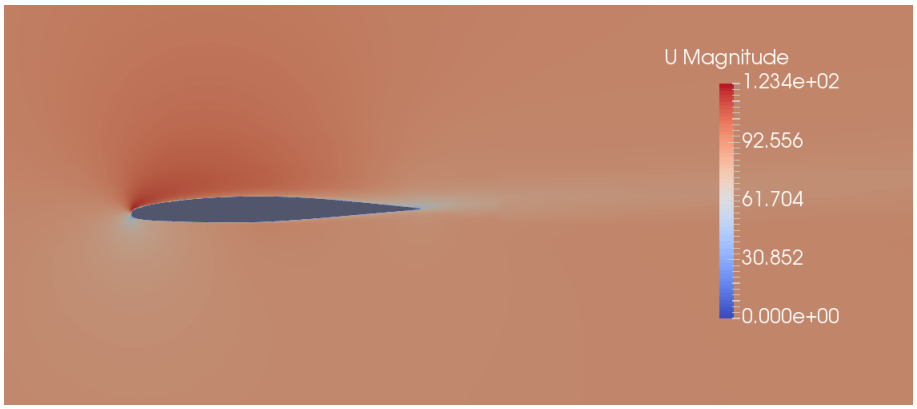
With a *simpleFoam* solution as the initial conditions, *rhoPimpleFoam is used* to calculate a transient solution. The *PIMPLE* algorithm is a combination of the *SIMPLE* and the *PISO* algorithms. The *PISO* algorithm only differs slightly from the *SIMPLE* algorithm, with the two differences being that no under-relaxation is applied, and that the momentum corrector step is performed more than once. In the *SIMPLE* algorithm this is done until the solution is converged, but for the *PISO* algorithm, only a couple of times before the next time step is calculated. The procedure is printed here, with the differences expressed in bold:

1. Set the boundary conditions.
2. Solve the discretized momentum equation to compute the intermediate velocity field.
3. Compute the mass fluxes at the cells faces.
4. **Solve the pressure equation.**
5. Correct the mass fluxes at the cell faces.
6. Correct the velocities on the basis of the new pressure field.
7. Update the boundary conditions.
8. **Repeat from 3 for the prescribed number of times.**
9. **Increase the time step and repeat from 1.**

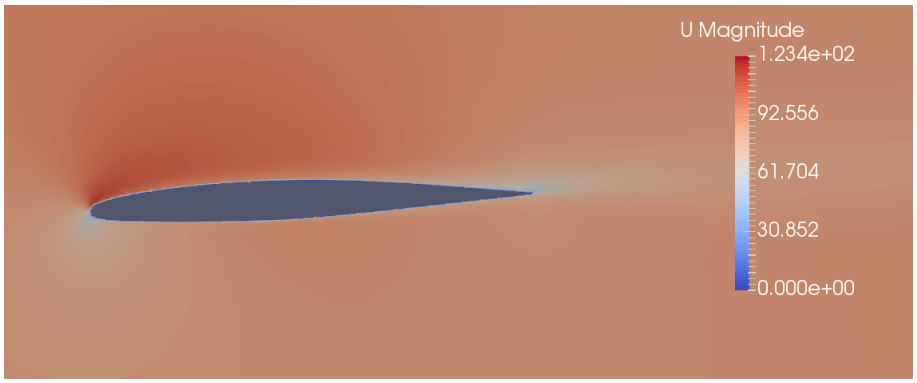
3.2.1 Results

The results of the rhoPimpleFoam simulations are shown here. Only plots and figures of Condition 2 are shown; for Condition 1 these can be found in appendix A.

In figure 5.1, two plots of the velocity are shown for two positions on the airfoil. One is 56 cm from the root of the airfoil and the second is at 112 cm from the root. The results are as expected. Around the whole airfoil, the velocity is 0 at the wall, obeying the no-slip condition. Also, the highest velocities are attained at the top of the airfoil, with values which lie well above the free stream velocity of 90 m/s.



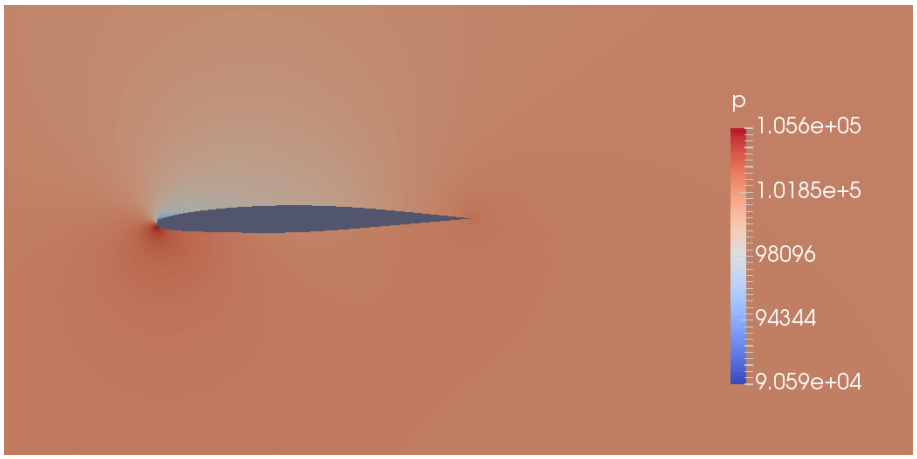
(a) At 56 cm from root



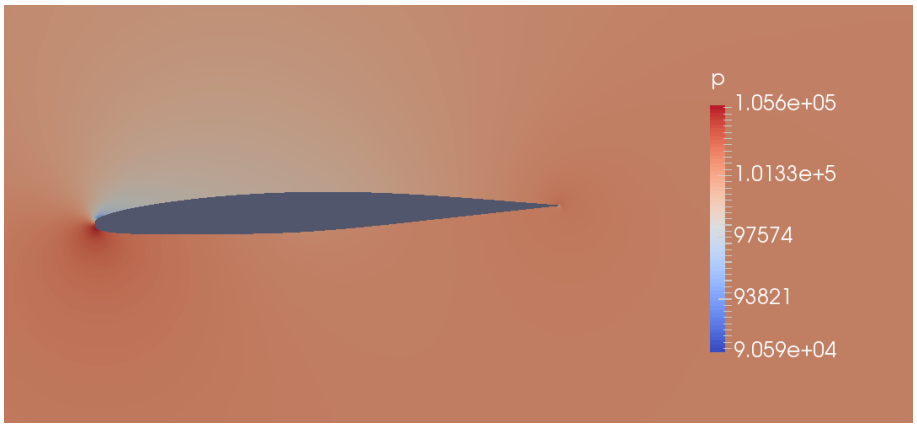
(b) At 112 cm from root

Figure 3.1: *Velocity plots at two locations on the airfoil for* $\alpha = 6^\circ$

In figure 5.2, two pressure plots are presented. Again, two locations on the airfoil are considered. The pressure plots are not surprising either. A stagnation point can be found at the leading edge of the airfoil, while the lowest pressures in the domain can be observed on the top of the airfoil. The lowest attained pressure in the system is well below the ambient pressure of 101325 Pa, while the highest pressures are bigger. This is the expected result.



(a) At 56 cm from root



(b) At 112 cm from root

Figure 3.2: *Pressure plots at two locations on the airfoil for* $\alpha = 6^\circ$

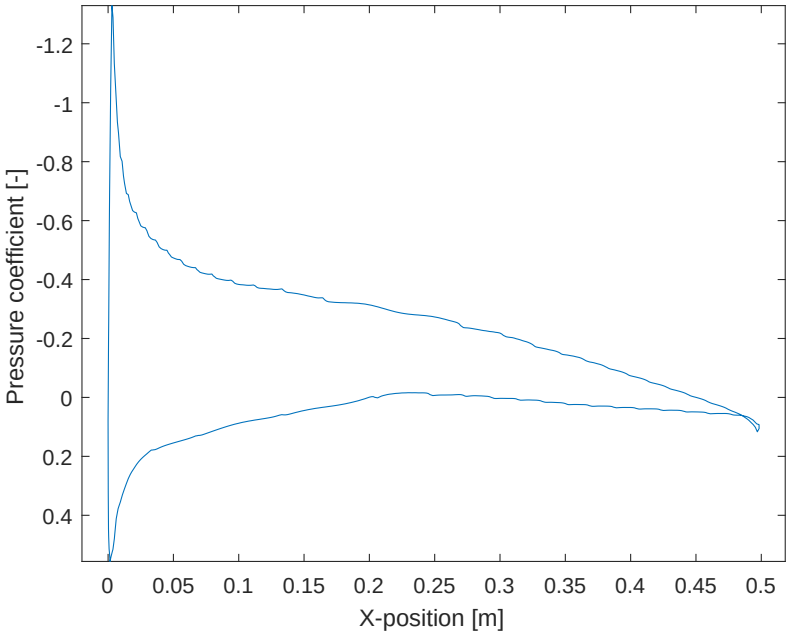
As a final check for the aerodynamics around the wing, the pressure coefficient around the airfoil at the two wing sections are considered. The pressure coefficient is the dimensionless pressure and is defined as:

$$C_p = \frac{p - p_\infty}{\frac{1}{2}\rho_\infty V_\infty^2} \tag{3.8}$$

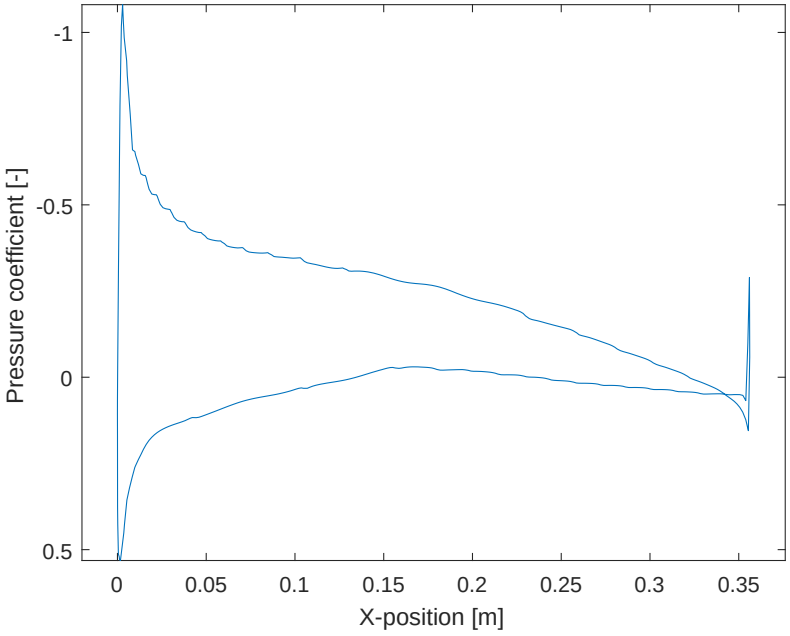
The plots are presented in figure 5.3 and some expected, but also some unexpected behaviour can be observed. The general shape of the curves is as expected, with a clear stagnation point and decreasing pressures when deviating from this point. Also, the pressure is lower on top of the airfoil as compared to the bottom of the airfoil. The curves are a bit wiggly, but this is probably because of the discretization on the wing.

As for the unexpected behavior, first of all the pressure coefficient at the stagnation point should be equal to unity, which it is not for both positions on the wing. Secondly, at the trailing edge the line crosses itself and in figure 5.3 (b), even a strange peak in the pressure coefficient can be observed. This is a highly unphysical phenomenon of which the origins are unknown. Because for impingement on the wing the pressure at the trailing edge is of minor importance, this is ignored.

The results for Condition 1 are shown in appendix A. The results are very similar, with only quantitative differences as expected. Unfortunately also for this case, the pressure coefficient plots show anomalies at the trailing edge. This indicates that the problem lies with the mesh and not with something else.



(a) Cp at 56 cm from the root



(b) Cp at 112 cm from the root

Figure 3.3: *Plots of the pressure coefficient along the airfoil at* $\alpha = 6^{\circ}$

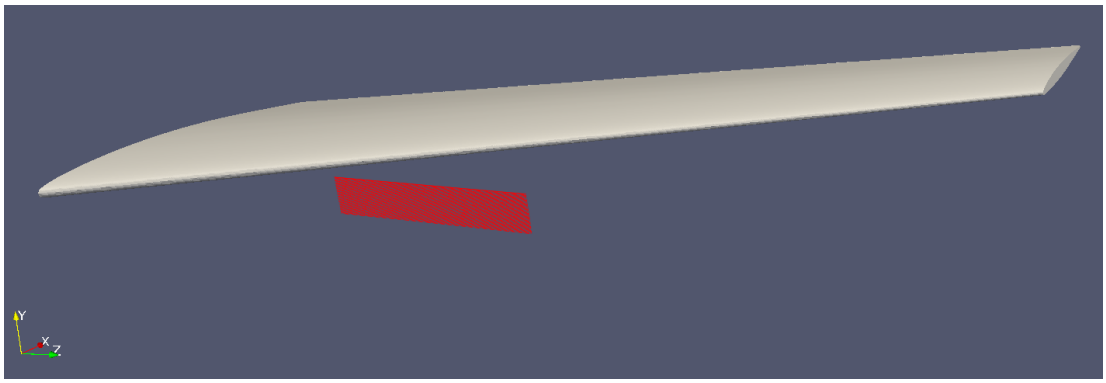
4 Impingement

In this chapter the simulations used to calculate droplet impingement on a 3D airfoil are considered.

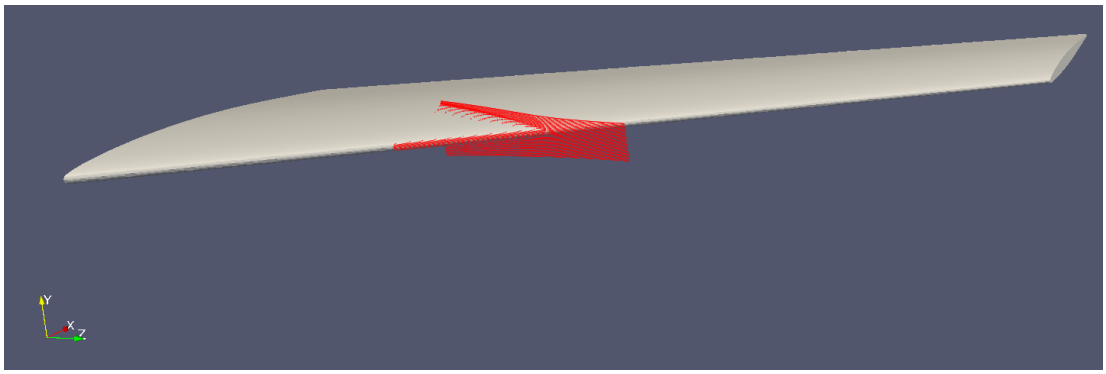
4.1 uncoupledKinematicParcelFoam

To calculate how water droplets impinge on the surface of the airfoil, the OpenFOAM solver *uncoupledKinematicParcelFoam* is used. This is a transient solver that uses a Langrangian approach to calculate particle (or kinematic parcels, hence the name) trajectories. Any amount of particles can be injected into the system, with a given location and velocity. Size and mass distributions are calculated by the solver for a given density, total mass of the particles and amount of particles. An additional file is used to specify the exact location of each of the particles, so that a cloud can be simulated. The input parameters for our case are specified in table 1.1. The input positions and velocity of the parcels is chosen so that the particles start just in front of the airfoil, at the same speed as was used as input when computing the flow field. The term 'uncoupled' in the solver's name refers to the fact that the parcels are assumed to have no significant effect on the flow field. Therefore, a constant flow field is assumed for the entire simulation and the one obtained using *rhoPimpleFoam* is therefore used.

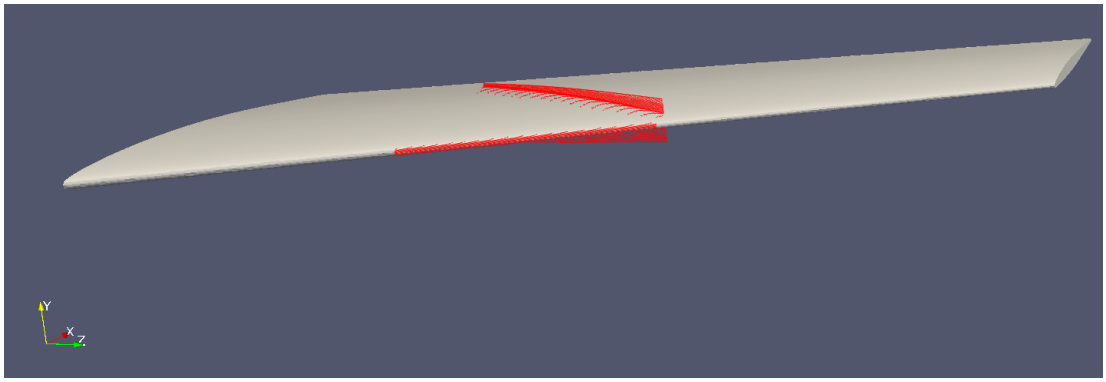
Due to time restrictions, only the case with an angle of attack of 6° was run, of which the results are presented here. A small 2D cloud was used to simulate the impingement on a small part of the airfoil. Either using a 3D cloud or the entire span of the airfoil would be too computationally heavy considering the resources available. In figure 4.1(a), the 2D cloud can be seen in its initial position. In figures 4.1(b) and (c), it is shown how the particles impact on the airfoild. Upon contact, a 'stick' condition is applied, so that the particles remain on the position where they first hit the airfoil. This data can then be used as input for an icing solver where, depending on the conditions, they can either be made to freeze upon impact, or made to be part of a water film as runback water. However, this is not covered in this work.



(a) Initial position



(b) Cloud hitting the airfoil



(c) Final position

Figure 4.1: *Water droplets (red) hitting the airfoil, as computed by uncoupledKinematicParcelFoam*

4.2 Catching efficiency

How particles, or droplets, impact on a surface is generally expressed using the catching efficiency β. This is quite simply the ratio of the particle density in the undisturbed cloud and the particle density on the airfoil:

$$\beta = \frac{\rho_{cloud}}{\rho_{surface}} \tag{4.1}$$

Impingement data is then collected by extracting how many particles hit each mesh face on the wing. By dividing the number of particles by the area of the individual face, the catching efficiency can be obtained. *OpenFOAM* does not assign face numbers in a structured manner, so it cannot be determined how the catching efficiency varies over the wing. It can be inspected what all the individual catching efficiencies are, but it is not possible to see on what location these cells are located. The values of β are expected

to lie between 0, for faces that are parallel to the flow or ones that are not hit at all, and 1, for faces perpendicular to the flow. Due to pressure and velocity differences near the wing, the maximum could become a little higher than 1. A histogram of all impingement values is shown in figure 4.2. It shows that the vast majority of all cells have a value between 0 and 2, which are considered reasonable. However, there is also a significant amount of mesh faces with a catching efficiency (much) higher than 2.

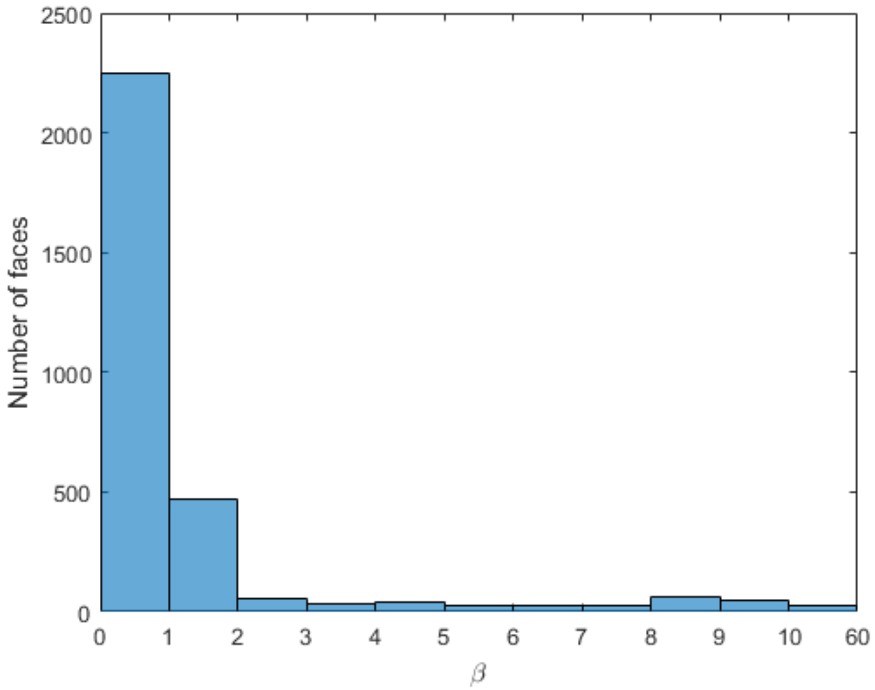


Figure 4.2: *Histogram of all mesh faces with impingement data*

4.2.1 Funnelling

The results with regard to the catching efficiency are unexpected, as the catching efficiency should take on a value between 0 and 1, but it is actually much higher than that. The average value of β turned out to be 1.66, with 13% of all mesh faces even having values above 2. The maximum catching efficiency value in the wing was even equal to 59(!). The root cause for this problem is unknown and should be investigated further. Although unexpected, this result was seen before at CIRA and is called funnelling, where somehow the particles happen to hit the airfoil closer to each other than they they started with respect to each other. The origin of this numerical problem is unknown, but it is very much unphysical. The most important impingement data is presented in table 4.1.

Parameter	**Value**
Average catching efficiency	1.66
Maximum catching efficiency	59.0
No. of faces with $0 < \beta < 2$	2720
No. of faces with $\beta > 2$	407
Percentage of faces with $\beta > 2$	13%

Table 4.1: *Catching efficiency data*

5 Conclusions and recommendations

In this work, the performance of the CFD package *OpenFOAM* was investigated regarding an icing analysis on a three dimensional wing: a GLC-305 airfoil which was swept, tapered and twisted from the root to the tip of the wing. Three analyses were performed: the meshing of the domain, the aerodynamics calculations, and the impingement. A thermodynamic analysis was not run due to time restrictions.

The meshing of the domain was performed using *snappyHexMesh*. It was found that this meshing tool is very sensitive to the input geometry and may perform poorly if a non-perfect geometry file is used. In the 2D case, this became apparent at the surface of the airfoil. At the location where a minor imperfection was present in the geometry, the added layers became distorted. In the 3D case, no layers were added by SHM at all. It was hypothesized that this was due to imperfections in the geometry triangulation as well. Without additional layers on the surface of the wing, the boundary conditions will not be resolved well. Especially for impingement this should not be ignored, because the area close to the airfoil is the region of interest and a perfect aerodynamic solution is wanted. In this work, no solution to the meshing problem was found, but it is advised that this problem is investigated further, or a mesh made with other software packages should be used. If one wants to use the meshing tool of *OpenFOAM* anyway, great care should be taken when selecting or constructing the geometry file to be used, as seemingly minor imperfection can have a big influence on the mesh quality.

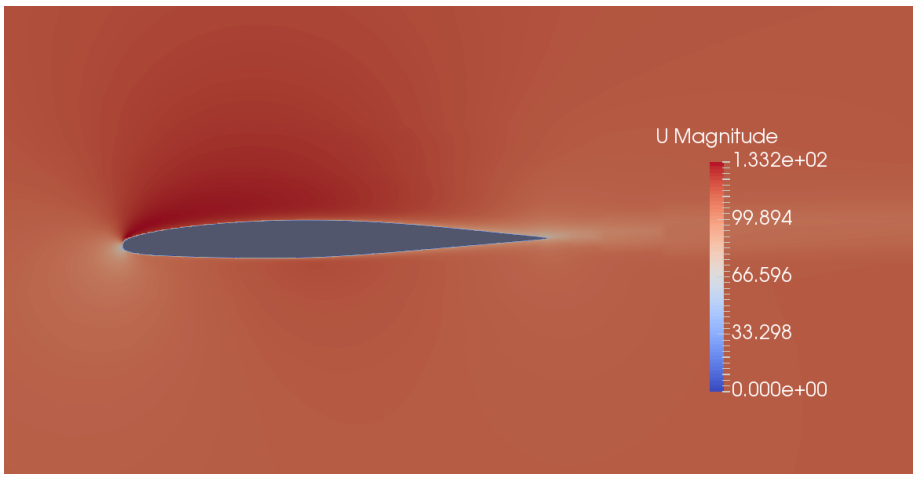
The aerodynamic computations were carried out using *simpleFoam* and *rhoPimpleFoam*. Both tools are easy to use and transferring the solution of simpleFoam as the initial condition in rhoPimpleFoam is very straightforward. The solutions converged quickly and the results looked very reasonable. The only anomaly occurred at the trailing edge of the wing, where some unexpected behaviour in the pressure was observed. It is hypothesized that the low quality mesh is causing this anomaly, but this is not confirmed. This issue was ignored in this work as the region of interest was located at the leading edge of the wing, but it is recommended to investigate the cause of the pressure error to exclude more fundamental problems with the aerodynamic solvers.

The impingement calculations were performed using uncoupledKinematicParcelFoam. This solver turned out take a lot of time to compute, but this is expected from a Lagrangian solver. Visually, the result looked reasonable. However, when looking more closely at the data, funnelling of the droplets was found. This indicates that an error is present in the solver itself, furthermore so because this was not the first time funnelling was observed at CIRA, and is therefore hard to solve. More investigation is definitely needed to get a reliable impingement result, which is needed for an actual icing analysis.

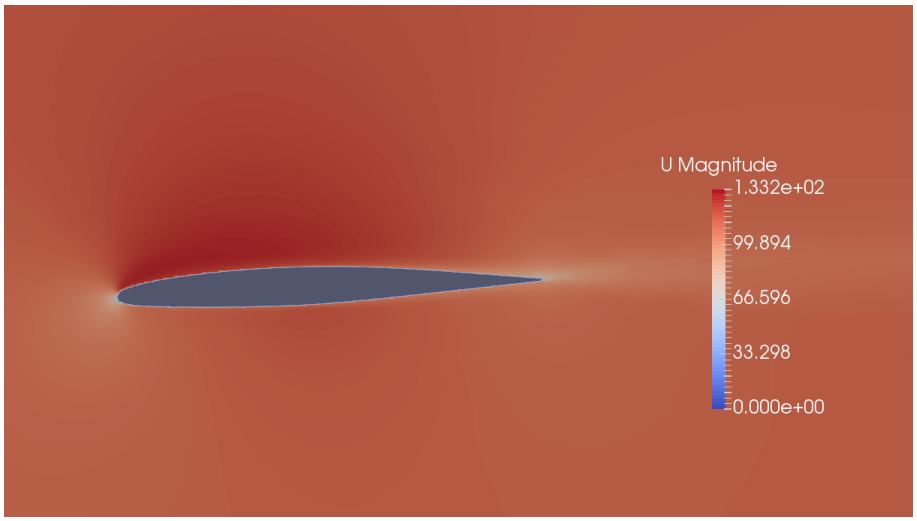
The CFD package used was *OpenFOAM*, an open source CFD solution with many different solvers. The advantages consist of the fact that open source software is free and freely adaptable, so that solvers can be modified easily to suit one's needs. This, however, also makes it prone to mistakes. This goes for the general settings as well. As everything is specified in simple text files, mistakes are easily made which can lead to crashing solvers or even worse, unphysical solutions. Also, very few documentation is available, making it very inaccessible to inexperienced users.

In terms of icing, in conclusion it can be stated that to this day, icing on aircraft wings remains an interesting subject, both from aeronautical point of view, as well as from numerical point of view. Especially in 3D, it remains difficult to obtain reliable part solutions. Although no actual icing simulations were performed in this work, the impingement proved to be very challenging already and the found results are questionable at best.

Appendix A - additional data

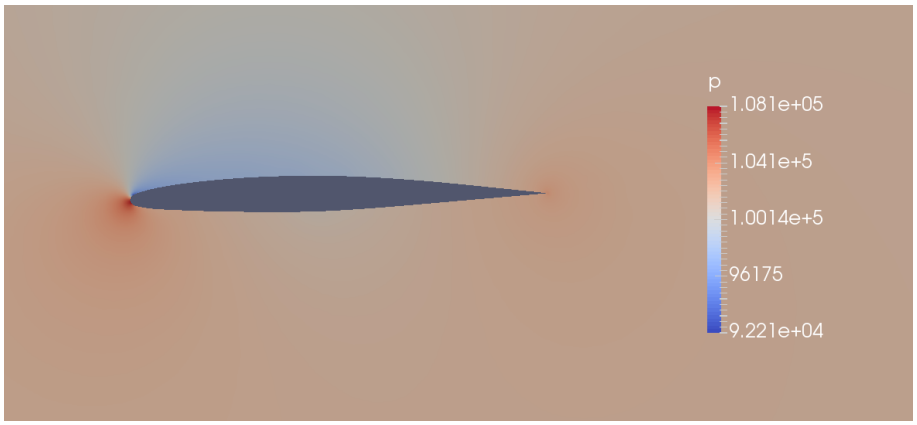


(a) At 56 cm from root

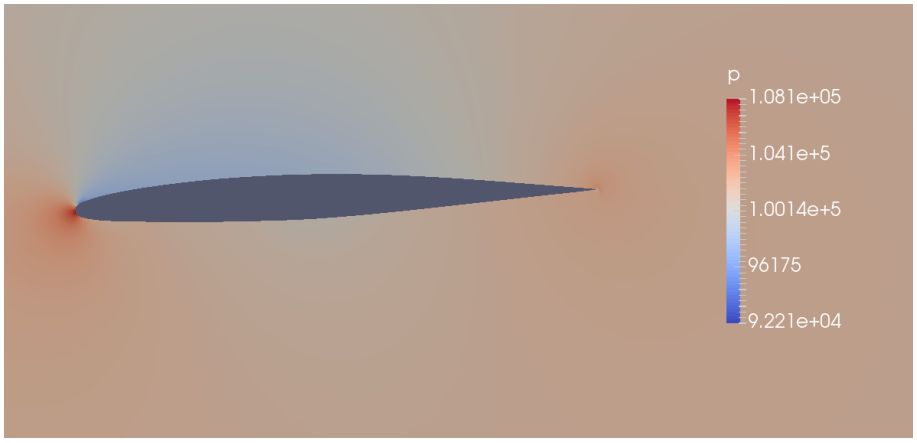


(b) At 112 cm from root

Figure 5.1: *Velocity plots at two locations on the airfoil for* $\alpha = 4^{\circ}$

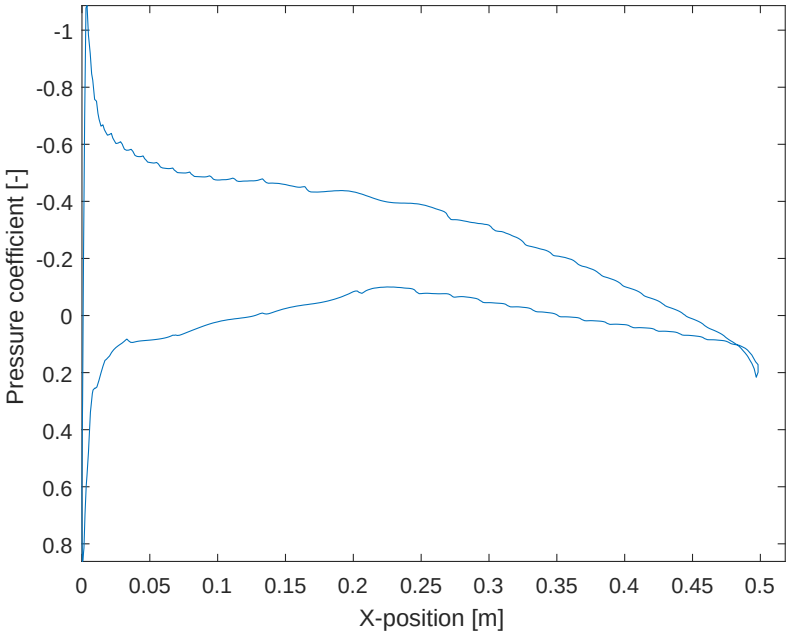


(a) At 56 cm from root

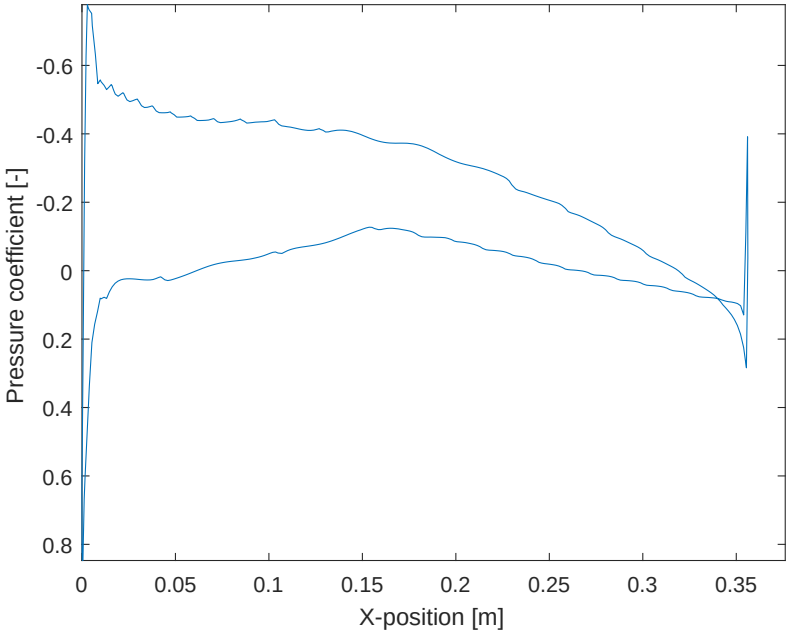


(b) At 112 cm from root

Figure 5.2: *Pressure plots at two locations on the airfoil for* $\alpha = 4^{\circ}$



(a) Cp at 56 cm from the root



(b) Cp at 112 cm from the root

Figure 5.3: *Plots of the pressure coefficient along the airfoil at* $\alpha = 4^{\circ}$

Bibliography

[1] Addy, H., Potatpczuk Jr., M., and Sheldon, D. (1997). Modern airfoil ice accretions. *AIAA Meeting Papers on Disc*.

[2] Authority, C. A. (2000). Aircraft icing handbook.

[3] Beld, E. J. (2013). Droplet impingement and film layer modeling as a basis for aircraft icing sumulations in openfoam.

[4] Foundation, A. A. S. (2008). Aircraft icing.

[5] Greenshields, C. (2015). Openfoam user guide.

[6] Myers, T., Charpin, J., and Thompson, C. (2002). Slowly accreting ice due to supercooled water impacting on a cold surface. *Physics of Fluids*, 14(1):240–256.

[7] OpenFOAM (2016). Unofficial openfoam wiki.

[8] Vargas, M., Papadakis, M., Potapczuk, M., Addy, H., Sheldon, D., and Giriunas, J. (2002). Ice accretions on a swept glc-305 airfoil. *NASA*.